Ghost Flag Writeup - CSCG 2023

This writeup explains how I solved the Ghost Flag challenge during the CSCG qualifiers of 2023.

- **Name:** Ghost Flag
- **Category:** Forensics
- **Difficulty:** Easy
- **Flag:** `CSCG{d3l3t3d_fl4g}`
- **Description:** You got access to a secret flag server, but can you find the flag?

In this challenge, we are given access to a Linux shell. We are given no other information, so the goal of the challenge is to figure out where the flag is stored, and how we can read it.

Background

To solve this challenge, we use the following commands:

- **ls:** Lists the contents of a directory. If we provide the **-a** flag, hidden files are shown as well.
- **cat:** Prints the content of one or more files.
- **grep:** Filters lines that match a certain pattern.
- **ps:** Shows information about active processes. If we provide the **a** and **x** flags, all processes on the system are displayed.
- **dd:** Copies a file. We can copy a specific part of the file with the **skip** and **count** arguments.

Furthermore, it is important to know that Linux provides more information about active processes, such as the list of mapped memory regions, in the `/proc` directory of the filesystem.

Solution

The shell initially starts in the directory `/home/ctf`. This directory appears to be empty, but `ls -a` shows that it contains some hidden files:

```
ctf@ghost-flag-vmtiqmuerx:/home/ctf$ ls -a
.  ..  .bash_history  .bash_logout  .bashrc  .flag.swp  .local  .profile
```

One of them looks interesting: `.flag.swp`. We can print its content with `cat`:

```
ctf@ghost-flag-vmtiqmuerx:/home/ctf$ cat .flag.swp
b0nano 6.2
```

Nano is a simple command line text editor. I could not find any mention of `b0nano` on Google, but Googling for `nano swap file` revealed that Nano creates a `.swp` file when a file is being edited, so that it can tell whether the file is currently being edited by another Nano process [1].

We list the active process with `ps ax` and indeed, the file `/home/ctf/flag` is being edited with Nano. This file does not exist in the filesystem, so we assume that we have to figure out what the user has written into the editor.

```
ctf@ghost-flag-kllqmlmfdt:/home/ctf$ ps ax
    PID TTY      STAT   TIME COMMAND
      1 ?        Ss     0:00 socat tcp-l:1024,reuseaddr,fork EXEC:/bin/bash,SU=c
      7 ?        S      0:00 socat EXEC:/usr/bin/nano /home/ctf/flag,SU=ctf,pty
     10 ?        S      0:00 /usr/bin/nano /home/ctf/flag
```

```
      49 ?        S      0:00 socat tcp-l:1024,reuseaddr,fork EXEC:/bin/bash,SU=c
      50 pts/1    Ss     0:00 /bin/bash
      53 pts/1    R+     0:00 ps ax
```

Notice that the PID of the process is 10. We need this to access the process in the `/proc` filesystem. The content of the file is stored somewhere in the heap memory of the process. First, we read the memory map of the process from `/proc/10/maps`:

```
ctf@ghost-flag-vmtiqmuerx:/home/ctf$ cat /proc/10/maps
5619db0bf000-5619db0c4000 r--p 00000000 08:03 11141975                   /usr/bin/nano
5619db0c4000-5619db0f6000 r-xp 00005000 08:03 11141975                   /usr/bin/nano
5619db0f6000-5619db102000 r--p 00037000 08:03 11141975                   /usr/bin/nano
5619db103000-5619db104000 r--p 00043000 08:03 11141975                   /usr/bin/nano
5619db104000-5619db105000 rw-p 00044000 08:03 11141975                   /usr/bin/nano
5619db105000-5619db106000 rw-p 00000000 00:00 0
5619dbd38000-5619dbddd000 rw-p 00000000 00:00 0                          [heap]
...
```

We see that the heap memory goes from address `0x5619dbd38000` to `0x5619dbddd000`. We can read this part of the memory from `/proc/10/mem` with `dd`. Because we only want the flag, we filter the output with `grep`:

```
ctf@ghost-flag-vmtiqmuerx:/home/ctf$ dd bs=1 if=/proc/10/mem skip=$((0x5619dbd38000)) count=$((0xa5000)) \
    | grep -ao 'CSCG{.*}'

dd: /proc/10/mem: cannot skip to specified offset
CSCG{d3l3t3d_fl4g}
675840+0 records in
675840+0 records out
675840 bytes (676 kB, 660 KiB) copied, 0.98462 s, 686 kB/s
```

We found the flag.

Mitigation

Apparently, a different person is working with sensitive data on the same user account as us. It is important to keep in mind that users can read the memory of all processes that are owned by them, even if the processes are created on a different terminal. Because the Nano process was owned by us, we could read its memory, and figure out what the person was typing. The solution is simple: on a shared machine, every person should be given their own user account. Do not provide a single user to multiple persons. Administrators should never log in on the user account of a customer. System processes should not be owned by the user account of a customer.

One user = one person.