

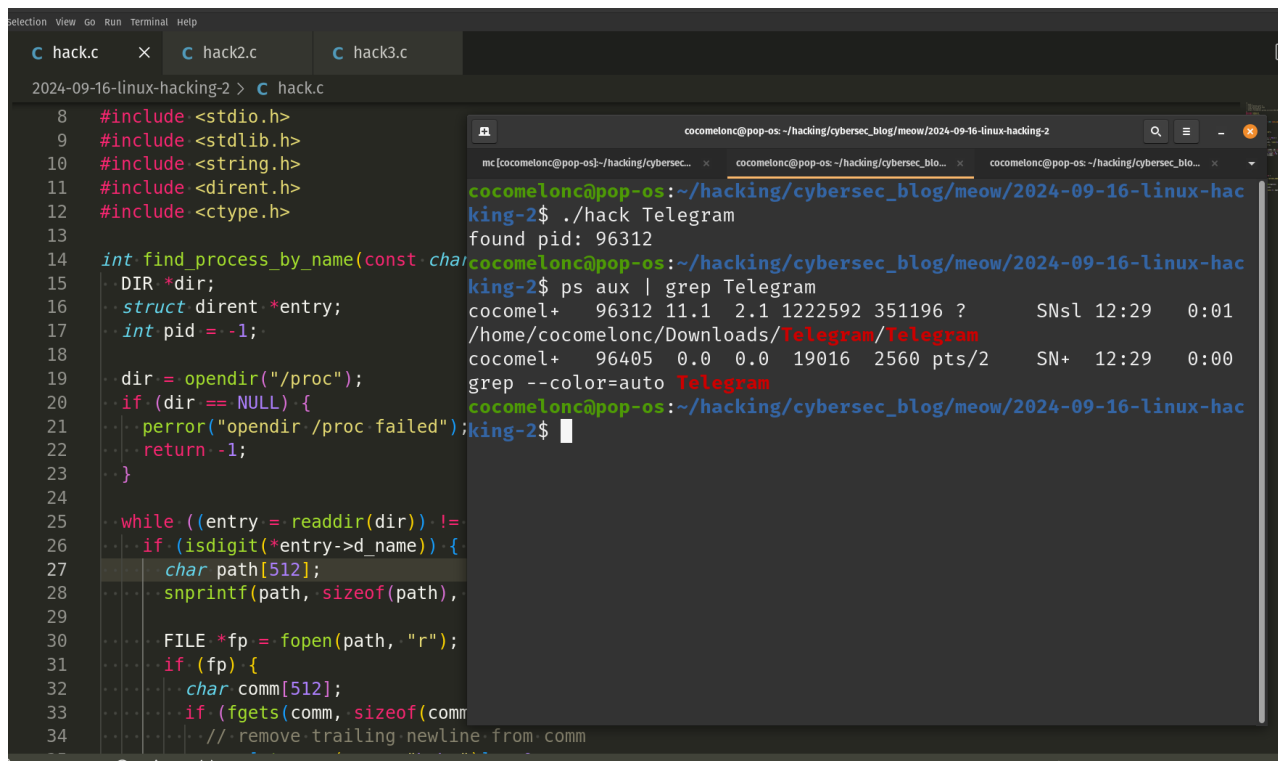
Linux malware development 2: find process ID by name. Simple C example.

cocomelonc.github.io/linux/2024/09/16/linux-hacking-2.html

September 16, 2024

5 minute read

Hello, cybersecurity enthusiasts and white hackers!



The screenshot shows a terminal window with a C program named 'hack.c' and its execution output. The C code defines a function 'find_process_by_name' that iterates through the '/proc' directory to find a process by name. The terminal output shows the command './hack Telegram' being executed, which returns the PID 96312. A subsequent 'ps aux | grep Telegram' command shows the process details for the Telegram application.

```
8 #include <stdio.h>
9 #include <stdlib.h>
10 #include <string.h>
11 #include <dirent.h>
12 #include <ctype.h>
13
14 int find_process_by_name(const char *name) {
15     DIR *dir;
16     struct dirent *entry;
17     int pid = -1;
18
19     dir = opendir("/proc");
20     if (dir == NULL) {
21         perror("opendir /proc failed");
22         return -1;
23     }
24
25     while ((entry = readdir(dir)) != NULL) {
26         if (isdigit(*entry->d_name)) {
27             char path[512];
28             snprintf(path, sizeof(path), "/proc/%s", entry->d_name);
29
30             FILE *fp = fopen(path, "r");
31             if (fp) {
32                 char comm[512];
33                 if (fgets(comm, sizeof(comm), fp) != NULL) {
34                     // remove trailing newline from comm
```

```
cocomelonc@pop-os: ~/hacking/cybersec_blog/meow/2024-09-16-linux-hacking-2
king-2$ ./hack Telegram
found pid: 96312
cocomelonc@pop-os: ~/hacking/cybersec_blog/meow/2024-09-16-linux-hacking-2
king-2$ ps aux | grep Telegram
cocomel+ 96312 11.1 2.1 1222592 351196 ?        Sns1 12:29   0:01
/home/cocomelonc/Downloads/Telegram/Telegram
cocomel+ 96405 0.0 0.0 19016 2560 pts/2    SN+ 12:29   0:00
grep --color=auto Telegram
cocomelonc@pop-os: ~/hacking/cybersec_blog/meow/2024-09-16-linux-hacking-2
king-2$
```

I promised to shed light on programming rootkits and other interesting and evil things when programming malware for Linux, but before we start, let's try to do simple things. Some of my readers have no idea how to do, for example, code injections into Linux processes.

Those who have been reading me for a very long time remember such an interesting and simple [example of finding the process identifier in Windows](#) for injection purposes.

practical example

Let's implement similar logic for Linux. Everything is very simple:

```

/*
 * hack.c
 * linux hacking part 2:
 * find process ID by name
 * author @cocomelonc
 * https://cocomelonc.github.io/linux/2024/09/16/linux-hacking-2.html
 */
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <dirent.h>
#include <ctype.h>

int find_process_by_name(const char *proc_name) {
    DIR *dir;
    struct dirent *entry;
    int pid = -1;

    dir = opendir("/proc");
    if (dir == NULL) {
        perror("opendir /proc failed");
        return -1;
    }

    while ((entry = readdir(dir)) != NULL) {
        if (isdigit(*entry->d_name)) {
            char path[512];
            snprintf(path, sizeof(path), "/proc/%s/comm", entry->d_name);

            FILE *fp = fopen(path, "r");
            if (fp) {
                char comm[512];
                if (fgets(comm, sizeof(comm), fp) != NULL) {
                    // remove trailing newline from comm
                    comm[strcspn(comm, "\r\n")] = 0;
                    if (strcmp(comm, proc_name) == 0) {
                        pid = atoi(entry->d_name);
                        fclose(fp);
                        break;
                    }
                }
                fclose(fp);
            }
        }
    }

    closedir(dir);
    return pid;
}

int main(int argc, char *argv[]) {
    if (argc != 2) {

```

```

    fprintf(stderr, "usage: %s <process_name>\n", argv[0]);
    return 1;
}

int pid = find_process_by_name(argv[1]);
if (pid != -1) {
    printf("found pid: %d\n", pid);
} else {
    printf("process '%s' not found.\n", argv[1]);
}

return 0;
}

```

My code demonstrates how to search for a running process by its name in Linux by scanning the `/proc` directory. It reads the process names stored in `/proc/[pid]/comm`, and if it finds a match, it retrieves the process ID (`PID`) of the target process.

As you can see there are only two functions here. First of all, we implemented `find_process_by_name` function. This function is responsible for searching for the process by name within the `/proc` directory.

It takes a process name (`proc_name`) as input and returns the `PID` of the found process or `-1` if the process is not found.

The function uses the `opendir()` function to open the `/proc` directory. This directory contains information about running processes, with each subdirectory named after a process ID (`PID`).

Then, iterate through entries in `/proc`:

```
while ((entry = readdir(dir)) != NULL) {
```

the `readdir()` function is used to iterate through all entries in the `/proc` directory, each entry represents either a running process (if the entry name is a number) or other system files.

Then checks whether the entry name represents a number (i.e., a process ID). Only directories named with digits are valid process directories in `/proc`:

```
if (isdigit(*entry->d_name)) {
```

Note that, the `comm` file inside each `/proc/[pid]` directory contains the name of the executable associated with that process:

```
snprintf(path, sizeof(path), "/proc/%s/comm", entry->d_name);
```

that means, we constructs the full path to the `comm` file by combining `/proc/`, the process ID (`d_name`), and `/comm`.

Finally, we open `comm` file, read process name and compare it:

```
FILE *fp = fopen(path, "r");
if (fp) {
    char comm[512];
    if (fgets(comm, sizeof(comm), fp) != NULL) {
        // remove trailing newline from comm
        comm[strcspn(comm, "\r\n")] = 0;
        if (strcmp(comm, proc_name) == 0) {
            pid = atoi(entry->d_name);
            fclose(fp);
            break;
        }
    }
}
```

Then, of course, close the directory and return.

The second function is the `main` function:

```
int main(int argc, char *argv[]) {
    if (argc != 2) {
        fprintf(stderr, "usage: %s <process_name>\n", argv[0]);
        return 1;
    }

    int pid = find_process_by_name(argv[1]);
    if (pid != -1) {
        printf("found pid: %d\n", pid);
    } else {
        printf("process '%s' not found.\n", argv[1]);
    }

    return 0;
}
```

Just check command-line args and run process finding logic.

demo

Let's check everything in action. Compile it:

```
gcc -z execstack hack.c -o hack
```

```

cocomelonc@pop-os: ~/hacking/cybersec_blog/meow/2024-09-16-linux-hacking-2
cocomelonc@pop-os:~/hacking/cybersec_blog/meow/2024-09-16-linux-hacking-2$ gcc -z execstack hack.c -o hack
cocomelonc@pop-os:~/hacking/cybersec_blog/meow/2024-09-16-linux-hacking-2$ ls -lt
total 24
-rwxrwxr-x 1 cocomelonc cocomelonc 16648 Sep 16 17:19 hack
-rw-rw-r-- 1 cocomelonc cocomelonc 1231 Aug 14 10:44 hack.c
cocomelonc@pop-os:~/hacking/cybersec_blog/meow/2024-09-16-linux-hacking-2$

```

Then run it in linux machine:

`.\hack [process_name]`

```

cocomelonc@pop-os:~/hacking/cybersec_blog/meow$ ps aux | grep Telegram
cocomel+ 75678 1.7 2.4 1318932 400036 ? Ssl 17:25 0:09 /home/cocomelonc/Downloads/Telegram/Telegram
cocomel+ 76342 0.0 0.0 19016 2432 pts/2 SN+ 17:34 0:00 grep --color=auto Telegram
cocomelonc@pop-os:~/hacking/cybersec_blog/meow$ cd 2024-09-16-linux-hacking-2/
cocomelonc@pop-os:~/hacking/cybersec_blog/meow/2024-09-16-linux-hacking-2$ ./hack Telegram
found pid: 75678
cocomelonc@pop-os:~/hacking/cybersec_blog/meow/2024-09-16-linux-hacking-2$

```

```

C hack.c u x
2024-09-16-linux-hacking-2 > C hack.c
7 int find_process_by_name(const char *proc_name) {
39
40     closedir(dir);
41     return pid;
42 }
43
44 int main(int argc, char *argv[]) {
45     if (argc != 2) {
46         fprintf(stderr, "usage: %s <proc_name>\n", argv[0]);
47         return 1;
48     }
49
50     int pid = find_process_by_name(argv[1]);
51     if (pid != -1) {
52         printf("found pid: %d\n", pid);
53     } else {
54         printf("process '%s' not found.\n", argv[1]);
55     }
56
57     return 0;
58 }

```

As you can see, everything is wokred perfectly. We found Telegram ID (75678) in my case!
 =^..^=

It all seems very easy, doesn't it?

But there is a caveat. If we try to run it for processes like `firefox` in my example:

`.\hack firefox`

we got:

```
cocomelonc@pop-os:~/hacking/cybersec_blog/meow/2024-09-16-linux-hacking-2$ ./hack firefox
process 'firefox' not found.
cocomelonc@pop-os:~/hacking/cybersec_blog/meow/2024-09-16-linux-hacking-2$ ps aux | grep firefox
cocomel+    2586  3.8  4.3 13462428 695296 ?        Sl    Sep13 160:33
firefox
cocomel+    3239  0.0  0.1 279896 25344 ?          S    Sep13   0:01
/usr/lib/firefox/firefox-bin -contentproc -greomni /usr/lib/firefox
/omni.ja -appomni /usr/lib/firefox/browser/omni.ja -appDir /usr/lib
/firefox/browser {1e774095-c604-40a3-ae65-d34c473e348c} 2586 true f
orkserver
cocomel+    3243  0.0  0.2 294980 33788 ?          Sl    Sep13   0:01
/usr/lib/firefox/firefox-bin -contentproc -parentBuildID 2024032123
```

The issue we're facing may stem from the fact that some processes, like `firefox`, might spawn child processes or multiple threads, which might not all use the `comm` file to store their process name.

The `/proc/[pid]/comm` file stores the executable name without the full path and may not reflect all instances of the process, especially if there are multiple threads or subprocesses under the same parent.

So possible issues in my opinion are:

- different process names in `/proc/[pid]/comm`: child processes or threads could use different naming conventions or might not be listed under `/proc/[pid]/comm` as `firefox`.
- zombies or orphan processes: some processes might not show up correctly if they are in a zombie or orphaned state.

practical example 2

Instead of reading the `comm` file, we can check the `/proc/[pid]/cmdline` file, which contains the full command used to start the process (including the process name, full path, and arguments). This file is more reliable for processes that spawn multiple instances like `firefox`.

For this reason I just created another version (`hack2.c`):

```

/*
 * hack2.c
 * linux hacking part 2:
 * find processes ID by name
 * author @cocomelonc
 * https://cocomelonc.github.io/linux/2024/09/16/linux-hacking-2.html
 */
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <dirent.h>
#include <ctype.h>

void find_processes_by_name(const char *proc_name) {
    DIR *dir;
    struct dirent *entry;
    int found = 0;

    dir = opendir("/proc");
    if (dir == NULL) {
        perror("opendir /proc failed");
        return;
    }

    while ((entry = readdir(dir)) != NULL) {
        if (isdigit(*entry->d_name)) {
            char path[512];
            snprintf(path, sizeof(path), "/proc/%s/cmdline", entry->d_name);

            FILE *fp = fopen(path, "r");
            if (fp) {
                char cmdline[512];
                if (fgets(cmdline, sizeof(cmdline), fp) != NULL) {
                    // command line arguments are separated by '\0', we only need the first
                    argument (the program name)
                    cmdline[strcspn(cmdline, "\0")] = 0;

                    // perform case-insensitive comparison of the base process name
                    const char *base_name = strrchr(cmdline, '/');
                    base_name = base_name ? base_name + 1 : cmdline;

                    if (strcasecmp(base_name, proc_name) == 0) {
                        printf("found process: %s with PID: %s\n", base_name, entry->d_name);
                        found = 1;
                    }
                }
                fclose(fp);
            }
        }
    }

    if (!found) {

```

```

    printf("no processes found with the name '%s'.\n", proc_name);
}

closedir(dir);
}

int main(int argc, char *argv[]) {
    if (argc != 2) {
        fprintf(stderr, "usage: %s <process_name>\n", argv[0]);
        return 1;
    }

    find_processes_by_name(argv[1]);

    return 0;
}

```

As you can see, this is an updated version of the code that reads from `/proc/[pid]/cmdline` instead.

But the file `/proc/[pid]/cmdline` or `/proc/[pid]/status` may not always show all subprocesses or threads correctly.

demo 2

Let's check second example in action. Compile it:

```
gcc -z execstack hack2.c -o hack2
```

The image shows a terminal window with the following output:

```

cocomelonc@pop-os:~/hacking/cybersec_blog/meow/2024-09-16-linux-hacking-2$ gcc -z execstack hack2.c -o hack2
cocomelonc@pop-os:~/hacking/cybersec_blog/meow/2024-09-16-linux-hacking-2$ ls -lt
total 48
-rwxrwxr-x 1 cocomelonc cocomelonc 16656 Sep 17 08:22 hack2
-rw-rw-r-- 1 cocomelonc cocomelonc 1640 Sep 17 08:15 hack2.c
-rw-rw-r-- 1 cocomelonc cocomelonc 1392 Sep 16 18:32 hack.c
-rwxrwxr-x 1 cocomelonc cocomelonc 16648 Sep 16 17:19 hack
cocomelonc@pop-os:~/hacking/cybersec_blog/meow/2024-09-16-linux-hacking-2$

```

Then run it in linux machine:

```
.\hack [process_name]
```



```

cocomelonc@pop-os:~/hacking/cybersec_blog/meow/2024-09-16-linux-hac
king-2$ ./hack2 firefox
found process: firefox with PID: 2586
cocomelonc@pop-os:~/hacking/cybersec_blog/meow/2024-09-16-linux-hac
king-2$ ./hack2 Firefox
found process: firefox with PID: 2586
cocomelonc@pop-os:~/hacking/cybersec_blog/meow/2024-09-16-linux-hac
king-2$ ps aux | grep firefox | head -n 3 | grep firefox
cocomel+  2586  3.3  4.5 13450656 736480 ?      Sl   Sep13 168:02
firefox
cocomel+  3239  0.0  0.1 279896 25344 ?        S    Sep13  0:01
/usr/lib/firefox/firefox-bin -contentproc -greomni /usr/lib/firefox
/omni.ja -appomni /usr/lib/firefox/browser/omni.ja -appDir /usr/lib
/firefox/browser {1e774095-c604-40a3-ae65-d34c473e348c} 2586 true f
orkserver

```

As you can see, it's correct.

I hope this post with practical example is useful for malware researchers, linux programmers and everyone who interested on linux kernel programming and code injection techniques.

[Find process ID by name. Windows version](#)
[source code in github](#)

| This is a practical case for educational purposes only.

Thanks for your time happy hacking and good bye!

PS. All drawings and screenshots are mine