CVE-2020-29664

CVE-2020-29664 is a security vulnerability discovered during our research, resulting in local arbitrary code execution without any hardware modifications on the DJI Mavic 2 Remote Controller and Leadcore processor on the DJI Mavic 2 Zoom drone.

During the firmware upgrade process a signature file (*file.cfg.sig*) and one or more firmware files (*file.cfg.fw*) get uploaded to the device. The signature file contains XML-data describing which firmware files have been uploaded and their corresponding meta-data such as file size and checksums. The program in charge of processing firmware upgrades is *dji_sys*.

```
int main() {
/* ... */
  snprintf(cmd,0x100,"busybox find %s" \
    "-name \"*.cfg.sig\"",
    "/data/upgrade/backup/");

  FILE *f = popen(cmd,"r");
  if (f)
    fgets(cfg_name, 0x100, f);
/* ... */
  int verify_ret = dji_verify_sig(\
    cfg_name,"/tmp/wm330_0000.xml",&DAT_0005c8a5);
/* ... */
}
```

Figure 1: Psuedocode of how signatures gets after upload.

Before the upgrade is performed, the signature and firmware files are verified to make sure that the data which was just uploaded came from DJI. The program branches out to the program *dji_verify* via the C programming language function popen. The program *dji_verify* verifies the file's signature to confirm the file sent is signed by DJI.

```
int dji_verify_sig(char *cfg_name,\
                   char *param_2, char *param_3) {
  /* ... */
  snprintf(cmd,0x100, \
    "dji_verify -n %s -o %s '%s'", \
    param_3,param_2,cfg_name);

  FILE *f = popen(cmd,"r");
  if (f)
    fgets(cmd_output,0x100,f);
  /* ... */
}
```

Figure 2: Psuedocode of how signatures gets verified.

The first parameter to this function is the filename of the signature file or firmware file that will be verified. This string gets used by the call to *snprintf*, which builds a shell-command executing *dji_verify* with a set of parameters, where the last parameter is the provided filename surrounded by apostrophes. This is done to prevent the filename from being interpreted as a valid shell-command syntax, which could allow an attacker to execute arbitrary shell-commands (command injection vulnerability).

```
The  popen() function opens a process by creating a pipe, forking,
and invoking the shell. Since a pipe is by definition unidirectional,
the type argument may specify only reading or writing, not both.
The resulting stream is correspondingly read-only or write-only.

The  command  argument is a pointer to a null-terminated string
containing a shell command line.
This command is passed to /bin/sh using the -c flag.
Interpretation, if any, is performed by the shell.
```

Figure 3: Manual page for the function popen

The method in which the filename argument is sanitised (via apostrophes) is flawed. By supplying a filename containing two apostrophes, it is possible to escape the previous apostrophes and execute arbitrary commands. We have verified this vulnerability and written a proof-of-concept capable of interacting with the device via a shell.

```
def hack_device(command):
  evil_name =  "wm240_rcAA'`"
  evil_name += command
  evil_name += "`'.cfg.sig"
  send_firmware_file(evil_name)
```

Figure 4: Psuedo code for how to execute arbitrary commands