Talos Vulnerability Report

TALOS-2021-1428

# Reolink RLC-410W "factory" binary firmware update vulnerability

JANUARY 26, 2022

CVE NUMBER

CVE-2021-40419

Summary

A firmware update vulnerability exists in the "factory" binary of reolink RLC-410W v3.0.0.136_20121102. A specially-crafted series of network requests can lead to arbitrary firmware update. An attacker can send a sequence of requests to trigger this vulnerability.

Tested Versions

Reolink RLC-410W v3.0.0.136_20121102

Product URLs

RLC-410W - https://reolink.com/us/product/rlc-410w/

CVSSv3 Score

10.0 - CVSS:3.0/AV:N/AC:L/PR:N/UI:N/S:C/C:H/I:H/A:H

CWE

CWE-489 - Leftover Debug Code

Details

The Reolink RLC-410W is a WiFi security camera. The camera includes motion detection functionalities and various methods to save the recordings.

The Reolink RLC-410W uses a binary called `factory`, which is allegedly a leftover debug binary. This binary allows to perform, without authentication, several critical operations. For example, the upload of the firmware and the factory reset of the device.

The `wait_for_connection` function listens for UDP packets on port 2009:

```
undefined4 wait_for_connection(void)

{
  [...]
  sockaddr.sin_zero._8_4_ = 1;
  sockaddr_len = 0x10;
  sockaddr.sa_family = 2;
  sockaddr._0_4_ = 0;
  sockaddr.sin_zero._12_4_ = 0;
  sockaddr.sin_addr = 0;
  sockaddr.sin_zero._0_4_ = 0;
  sockaddr.sin_zero._4_4_ = 0;
  sockaddr.sa_data._2_4_ = 0;
  sockaddr.sa_data._6_4_ = 0;
  sockaddr.sa_data._10_4_ = 0;
  sockaddr.sa_data._0_2_ = 55559;
  [...]
      __fd = socket(AF_INET,1,0);
      [...]
        iVar2 = bind(__fd,&sockaddr,0x10);
        [...]
        memset(recv_data,0,0x400);
        sVar3 = recvfrom(__fd,recv_data,0x400,0,(sockaddr *)&sockaddr,&sockaddr_len);
        if (sVar3 < 1) {
            [...]
        }
        else {
          if (recv_data._0_4_ == 0xfafafa) {                                                    [1]
            source_IP = inet_ntoa((in_addr)sockaddr.sin_addr);
            strncpy((char *)&IP_of_brodcaster,source_IP,0x17);
            IP_set = 1;
           [...]
            close(__fd);
            return 0;
          }
          [...]
}
```

At `[1]` it is checked if the received UDP packet contains, as first four bytes, the data `\xfa\xfa\xfa\x00`. If this is the case, the IP of the sender is saved and later used, by the `factory` binary, for connecting back to the sender on port 9123. With this connection, it is possible to interact with the `factory` binary and send several commands that the binary will execute.

The messages received by the `factory` binary, through the previously established connection, are elaborated and queued. Eventually, the received data reach the `manage_recv_queue` function. This function will iterate through the list of the queued messages and elaborate them.

The `manage_recv_queue` function:

```
undefined4 manage_recv_queue(factory *factory)

{
  [...]
  msg_cursor = (factory->msg_node).node_start;
  do {
    while( true ) {
      if (msg_cursor == (factory_msg_node *)&(factory->msg_node).node_end) {
        return 0;
      }
      factory_msg = msg_cursor->data;
      if (factory_msg->message_handling_status == UNHANDLED) break;
        [...]
    }
    if ((factory->is_update_preapre_ongoing == 0) || (factory_msg->message_cmd != 8)) {
      factory_msg->message_handling_status = HANDLING_WIP;
      [...]
        switch(factory_msg->message_cmd) {
        case 0:
          ret_code = get_Sp_data(factory,factory_msg);
          break;
        case 1:
          ret_code = sys_time_set(factory,factory_msg);
          break;
        case 2:
          ret_code = set_Sp_uid(factory,factory_msg);
          break;
        default:
          goto switchD_0040ec64_caseD_3;
        case 6:
          ret_code = check_if_filename_is_new_FW(factory,factory_msg);
          break;
        case 7:
          ret_code = check_if_filename_is_new_FW_2(factory,factory_msg);
          break;
        case 8:
          ret_code = update_command(factory,factory_msg);
          break;
        case 9:
          ret_code = maybe_sanity_check_command(factory,factory_msg);
          break;
        case 0xb:
          ret_code = factory_reset(factory,factory_msg);
          break;
        case 0xc:
          ret_code = probably_compare_uid(factory,factory_msg);
          break;
        case 0x11:
          ret_code = get_Sp_info_XML(factory,factory_msg);
          break;
        case 0x12:
          ret_code = speed_test_start(factory,factory_msg);
          break;
        case 0x13:
          factory_msg->message_handling_status = PROBABLY_COMPLETED;
          goto LAB_0040eec4;
        case 0x14:
          ret_code = speed_test_stop(factory,factory_msg);
          break;
        case 0x15:
          ret_code = speed_test_result_set(factory,factory_msg);
        }
    [...]
    }
    [...]
}
```

The `manage_recv_queue` will iterate through the message queue. If a message is `UNHANDLED`, the `manage_recv_queue` will execute the functionality specified in the message. The `factory` binary allows, without authentication, several critical functionalities: firmware update, device factory reset, setting the time and others. The `update_command` functionality will allow to upload a firmware file. If the uploaded firmware file passes a CRC check, the firmware will be updated.

The `factory` binary is, allegedly, a leftover debug binary and should not exist in a release version of the firmware.

### Timeline

2021-12-13 - Vendor Disclosure
2022-01-19 - Vendor Patched

2022-01-26 - Public Release

### CREDIT

Discovered by Francesco Benvenuto of Cisco Talos.

---