Talos Vulnerability Report

TALOS-2021-1424

# Reolink RLC-410W device network settings OS command injection vulnerabilities

JANUARY 26, 2022

### CVE NUMBER

CVE-2021-40407,CVE-2021-40408, CVE-2021-40409, CVE-2021-40410,CVE-2021-40411,CVE-2021-40412

### Summary

Multiple os command injection vulnerabilities exist in the device network settings functionality of reolink RLC-410W v3.0.0.136_20121102. A specially-crafted HTTP request can lead to arbitrary command execution. An attacker can send an HTTP request 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

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

### CWE

CWE-78 - Improper Neutralization of Special Elements used in an OS Command (‘OS Command Injection’)

### Details

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

The RLC-410W exposes some privileged APIs to setup different networking parameters, like: DDNS, DNS, IP, Hostname, etc.

The `device` program is responsible for taking the change, through the APIs, and applying those into the system. These change are implemented using OS commands. Because these inputs are not validated properly, this lead to OS command injections.

The `API` that allows the manipulation of the DDNS settings is `SetDdns`, and the payload for changing the data looks like:

```
[
    {
        "cmd": "SetDdns",
        "action": 0,
        "param": {
            "Ddns": {
                "domain": <DOMAIN>,
                "enable": <ENABLE>,
                "password": <PASSWORD>,
                "type": <TYPE>,
                "userName": <USERNAME>
            }
        }
    }
]
```

From the API program, this information will be saved into a settings file and eventually used by the `device` program. Specifically the function that applies the DDNS settings is `set_dds_config`:

```
undefined4 set_dds_config(ddns_data_struct *ddns)

{
  [...]

  memset(ddns_shell_command,0,0x100);
  memset(ddns_type,0,0x80);
  memset(username_and_password,0,0x400);
  memset(b64_encoded,0,0x400);
  if (ddns == (ddns_data_struct *)0x0) {
    return 0xffffffff;
  }
  type_of_ddns = ddns->type_of_ddns;
  if (type_of_ddns == 3) {
    [...]
GOTO_FORMAT:
    pip_system(ddns_format_string);
    ddns_format_string = "/mnt/tmp/ddns/ddns-config %s %s %s %s %d %s";
  }
  else {
    if (type_of_ddns != 0) {
      if (type_of_ddns != 1) {
        if (type_of_ddns == 4) {
          if (((ddns->domain[0] == 0) && (ddns->username[0] == 0)) && (ddns->password[0] == 0)) {
            return 0;
          }
          snprintf(username_and_password,0x3ff,"%s:%s",ddns->username,ddns->password);
          username_and_password_len = strlen(username_and_password);
          iVar1 = bencode(username_and_password,username_and_password_len,b64_encoded,0x400);
          if (iVar1 < 0) {
            return 0xffffffff;
          }
          strcpy(ddns_type,"dynupdate.no-ip.com");
          if (ddns->ddns_server[0] == 0) {
            strcpy((char *)ddns->ddns_server,"dynupdate.no-ip.com");
          }
          pip_system("rm /mnt/tmp/ddns/ddnsrun -f");
          pip_system("ln -s /mnt/tmp/ddns/ddnsrun.noip /mnt/tmp/ddns/ddnsrun");
          snprintf(ddns_shell_command,0x100,"/mnt/tmp/ddns/ddns-config %s %s %s %s %d %s %s ",
                   ddns->domain,ddns->username,ddns->password,ddns_type,1,ddns->ddns_server,
                   b64_encoded);                                                                    [1]
        }
        else {
          [...]
        }
        goto GOTO_SYSTEM;
      }
      if (((ddns->domain[0] == 0) && (ddns->username[0] == 0)) && (ddns->password[0] == 0)) {
        return 0;
      }
      strcpy(ddns_type,"members.dyndns.org");
      pip_system("rm /mnt/tmp/ddns/ddnsrun -f");
      ddns_format_string = "ln -s /mnt/tmp/ddns/ddnsrun.dyndns /mnt/tmp/ddns/ddnsrun";
      goto GOTO_FORMAT;
    }
    [...]
  }
  snprintf(ddns_shell_command,0x100,ddns_format_string,ddns->domain,ddns->username,ddns->password,
           ddns_type,1,ddns->ddns_server);                                                          [2]
GOTO_SYSTEM:
  if (ddns_shell_command[0] != '\0') {
    pip_system(ddns_shell_command);                                                                 [3]
  }
  pip_system("/mnt/tmp/ddns/ddns-start &");
  return 0;
}
```

The `pip_system` is not much different from the `system` function.

The OS command string for changing the DDNS setting is prepared at `[1]` or `[2]`, based the DDNS `type`, and then the prepared command is executed at `[3]`.

The `API` that allows for the manipulation of the DNS settings is `SetLocalLink`, and the payload for changing the data looks like:

```
[
    {
        "cmd": "SetLocalLink",
        "action": 0,
        "param": {
            "LocalLink": {
                "dns": {
                    "auto": 0,
                    "dns1": <DNS1>,
                    "dns2": <DNS2>
                },
                "mac": <MAC>,
                "static": {
                    "gateway": <GW_IP>,
                    "ip": <CAMERA_IP>,
                    "mask": <NETMASK>
                },
                "type": <TYPE>
            }
        }
    }
]
```

From the API program, this information will be saved into a settings file and eventually used by the `device` program. Specifically the function that applies the DNS settings is `bc_dns_set`:

```
int bc_dns_set(char *interface,dns_data *dns_data)

{
  int res;
  char set_dns_shell_command [260];

  if (*interface != '\0') {
    memset(set_dns_shell_command,0,0x100);
    if ((dns_data->dns1[0] == 0) && (dns_data->dns2[0] == 0)) {
      pip_system("echo > /etc/resolv.conf");
      res = 0;
    }
    else {
      snprintf(set_dns_shell_command,0x100,"echo \"nameserver %s\" > /etc/resolv.conf",dns_data->dns1); [4]
      res = pip_system(set_dns_shell_command);                                                          [5]
      if (res < 0) {
        res = -1;
      }
      else {
        snprintf(set_dns_shell_command,0x100,"echo \"nameserver %s\" >> /etc/resolv.conf",
                 dns_data->dns2);                                                                       [6]
        res = pip_system(set_dns_shell_command);                                                        [7]
        res = res >> 0x1f;
      }
    }
    return res;
  }
  return -1;
}
```

At `[4]` the OS command string for the `dns1` is prepared and then used at `[5]` to execute the command. At `[6]` the OS command string for the `dns2` is prepared and then used at `[7]` to execute the command.

The `API` that allows for the manipulation of the hostname is `SetDevName`, and the payload for changing the data looks like:

```
[
    {
        "cmd": "SetDevName",
        "action": 0,
        "param": {
            "DevName": {
                "name": <DEVICE_NAME>
            }
        }
    }
]
```

From the API program, this information will be saved into a settings file and eventually used by the `device` program. Specifically the function that applies the DNS settings is `bc_dhcpc_start`:

```
int bc_dhcpc_start(char *interface,char *devname)

{
  int is_error;
  char dhcp_client_shell_command [256];
  char requested_ip [16];

  is_error = -1;
  if ((*interface != '\0') && (*devname != '\0')) {
    requested_ip._0_4_ = 0;
    requested_ip._4_4_ = 0;
    requested_ip._8_4_ = 0;
    requested_ip._12_4_ = 0;
    memset(dhcp_client_shell_command,0,0x100);
    is_error = generate_IP(requested_ip);
    if (-1 < is_error) {
      snprintf(dhcp_client_shell_command,0x100,
               "udhcpc -i %s -x hostname:\'%s\' -p /var/run/udhcpc.pid -r %s &",interface,devname,
               requested_ip);                                                                       [8]
      is_error = pip_system(dhcp_client_shell_command);                                             [9]
      is_error = is_error >> 0x1f;
    }
  }
  return is_error;
}
```

At `[8]` the OS command string for using the DHCP and setting the hostname is prepared and then used at `[9]` to execute the command.

All three of the APIs, `SetDdns`, `SetLocalLink` and `SetDevName`, can be used to reach the `pip_system` and exploit OS command injections. Their details are described below in dedicated sections.

### CVE-2021-40407 - SetDdns - domain

At `[1]` or `[2]`, based on DDNS type, the `ddns->domain` variable, that has the value of the `domain` parameter provided through the `SetDdns` API, is not validated properly. This would lead to an OS command injection. The string formatted at `[1]` or `[2]` is then used as OS command at `[3]`.

### CVE-2021-40408 - SetDdns - userName

At `[1]` or `[2]`, based on DDNS type, the `ddns->username` variable, that has the value of the `userName` parameter provided through the `SetDdns` API, is not validated properly. This would lead to an OS command injection. The string formatted at `[1]` or `[2]` is then used as OS command at `[3]`.

### CVE-2021-40409 - SetDdns - password

At [1] or [2], based on DDNS type, the ddns->password variable, that has the value of the password parameter provided through the SetDdns API, is not validated properly. This would lead to an OS command injection. The string formatted at [1] or [2] is then used as OS command at [3].

### CVE-2021-40410 - SetLocalLink - dns1

At [4] the dns_data->dns1 variable, that has the value of the dns1 parameter provided through the SetLocalLink API, is not validated properly. This would lead to an OS command injection. The string formatted at [4] is then used as OS command at [5].

### CVE-2021-40411 - SetLocalLink - dns2

At [6] the dns_data->dns2 variable, that has the value of the dns2 parameter provided through the SetLocalLink API, is not validated properly. This would lead to an OS command injection. The string formatted at [6] is then used as OS command at [7].

### CVE-2021-40412 - SetDevName - name

At [8] the devname variable, that has the value of the name parameter provided through the SetDevName API, is not validated properly. This would lead to an OS command injection. The string formatted at [8] is then used as OS command at [9].

#### Timeline

2021-12-06 - Vendor Disclosure
2022-01-19 - Vendor Patched

2022-01-26 - Public Release

#### CREDIT

Discovered by Francesco Benvenuto of Cisco Talos.

---

VULNERABILITY REPORTS

PREVIOUS REPORT TALOS-2021-1423

NEXT REPORT TALOS-2021-1425