Automating Unsolicited Richard Pics; Pwning 60,000 Digital Picture Frames

November 26, 2022

SHARE

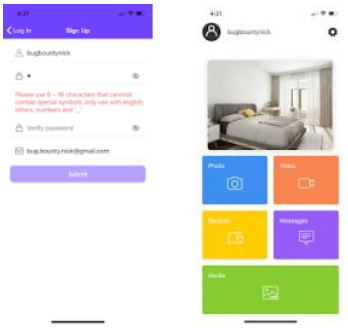
Summary The research for this post was done sometime in January of 2022, I was diagnosed with Cancer in February of 2022, and have been struggling to find to the time to finally post it. Much of it was written in chunks, so hopefully it makes sense. In this post I will talk about what led me down the path of researching the security of IoT digital picture frames. My research on these picture frames and supporting mobile application led to the creation of four CVE's. The vulnerabilities I discovered allowed me to access client information, clear-text credentials, bypass authentication and access controls of the frames.

Merry Christmas Mimi! During our annual brainstorming of Christmas gift ideas for my wife's elderly grandmother, we thought a digital picture frame would be perfect. Mimi is 94 years old, and lives in a local nursing home. Covid has made it difficult for family to visit her and for her to leave the home as much as she used to. We decided to look for a picture frame that would allow us to frequently send pictures of what the family is up to outside the walls of the nursing home. We researched different brands and functionality. We decided we wanted a wifi enabled picture frame, that would allow multiple family members to send photos to it.

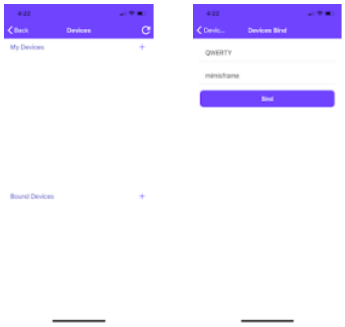
BIGASUO 10.1 Digital Picture Frame We looked on amazon to find a picture frame which was not super expensive, but also had good reviews. We settled on the BIGASUO 10.1 Digital Picture Frame. It has 371 reviews and nearly a 5-star rating. My wife purchased the frame and it arrived in a few days.

Configuring the frame Setting up the frame for mimi was very easy. The directions in

the setup guide ask that you download a mobile application called Ourphoto from the App store or Google play store. When you download the Ourphoto application you are asked to sign up for a new account. I noticed when filling out the form for an account that you were not allowed to use a complex password. This was a red-flag for me working in application security as my day job. I used a throw-away email account I didn't care about and decided to move forward with the setup.



The instructions then ask through the Ourphoto application to bind your frame to your account. This is done by entering the frame's unique device id along with a nickname for the frame.



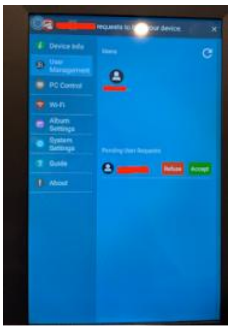
I setup different family members accounts on the frame so everyone could send photo's to mimi's frame at any time. I sent a few test photos, and so did the other members of the family. I really didn't spend much time at all exploring the device, as it had already been a few weeks after Christmas, and I had already dragged my feet on getting it setup. Everything seemed to be working properly. I packaged the frame up and sent it with my mother-in-law who was going to set it up at the nursing home. Overall, it was a painless process and the product worked at described.

Who is this person? This security research kicked off with a simple question from my mother-in-law. She sent a text to my wife that six pictures of a random women she has never seen before are on mimi's picture frame. My mother-in-law was concerned that the picture frame had been hacked, she deleted the photos then texted my wife. I asked her to disconnect the frame and drop it off so I could look at what was going on.

Not much information to go on When I got my hands on the picture frame, I started looking at the interface the end-user has access to. The touch-screen interface seems to be a cut-down, customized Android based operating system. The interface had a "Settings" area which I started to explore. The settings area offered "Device Info", "User Management", "PC control", "Wi-fi", "Album Settings", "System Settings", "Guide", and "About". I carefully reviewed each area to try to find an audit log or information

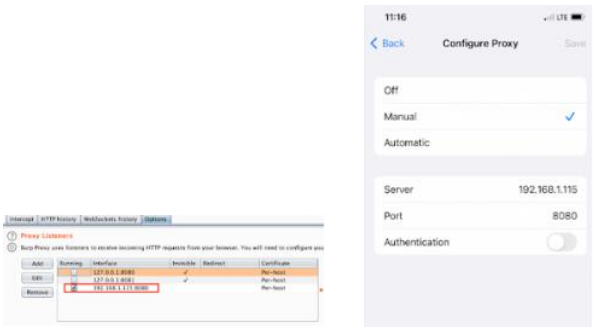
that could be useful to my research.

Focusing on the Settings area It was apparent that this picture frame was a blackbox with little to no troubleshooting or audit information to offer end-users. The "User Management" area showed all the family members that I had added to the picture frame, and a built-in administrator account that came on the picture frame. The process of adding a person requires that you bind your account in the Ourphoto mobile application to the picture frames unique frame id. Once a request has been made to bind to the picture frame, you must use the physical frame to accept the request through the touch screen. My current theory was that the built-in administrator account on the frame has sent the photos of the women in error and it was clearly some bug/glitch in the back-end api or infrastructure.

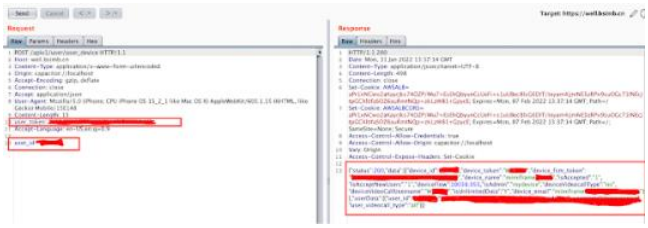


Time to roll up my sleeves As someone who works in the application security field, my toolbelt relies heavily on BurpSuite. For those who don't know what BurpSuite is, it's an interception proxy that allows someone to inspect the web traffic being sent and received by web browsers, mobile applications, and other web-based devices. My plan was to look at the Ourphoto mobile application to understand how it interacts with the back-end api and the picture frame itself.

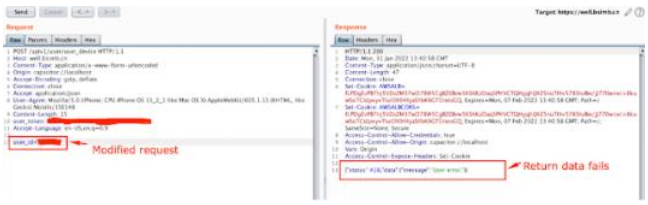
Broken access control everywhere…Ourphoto App has big issues The first part of my research focused on finding answers to my theory that the built-in admin account sent the unknown photos to the frame. I started inspecting the traffic from the Ourphoto mobile app by joining it the same wifi network as my laptop. I configured Burpsuite to listen on my wifi interface, and setup the iphone to proxy traffic through Burpsuite.



The first thing I noticed when running the Ourphoto application and inspecting the web requests was a call to the end-point /apiv1/user/user_device. The request specified a user_id parameter. The user_id parameter was the assigned unique identifier for the account I created with the Ourphoto app. There was also a user_token header, which appeared to be another unique identifier for the account I created in the Ourphoto app. The response to this request provided unique and sensitive information needed for my digital picture frame. This information device_id, device_token, device_fcm_token, device_name, device_email, username, and user_id. These fields are used to authenticate and authorize functionality within the digital picture frame.



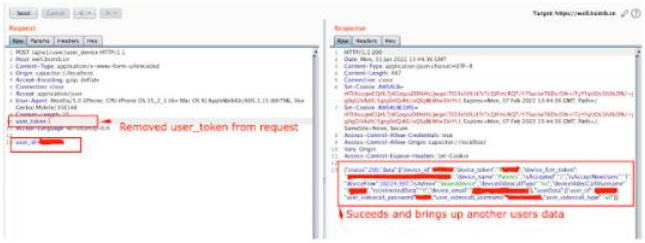
The first thing I checked was the presence of an Insecure Direct Object Reference (IDOR) with my user_id number. I took my unique number and decreased it by 1 number. For example, if my number was 123456, I changed it to 123455. This seemed to fail as I hoped it would.



I then focused in on that user_token header. I assumed the user_token header was being matched to the user_id parameter. If they didn't match for that user_id, then I would get the "User error". I decided to see what would happen if I removed the value

completely for the user_token header with my modified user_id (i.e. 123455). The result was success. The response pulls up another user's information. This means that the logic checked for a match with the user_token but failed open if the user_token

was blank. The user_id was a sequential number, so it meant I had access to the information for 60,000+ user accounts. This discovery actually lead to three vulnerabilities, Improper Access Control for the lack protection with the user_token and Insecure Direct Object Reference for the user_id and device_user parameters.

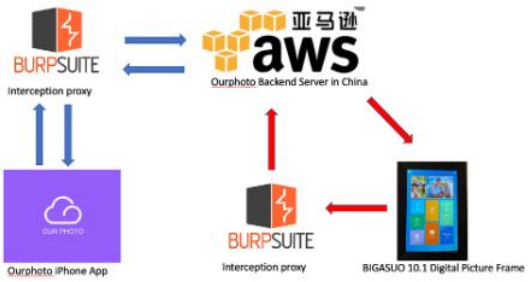


The Insecure Direct Object Reference vulnerability with the device_user parameter was found on a different end-point. The /device/device_user endpoint allowed me to enumerate other end-users digital picture frame information through the device_id parameter. This end-point revealed more sensitive information.

Time to make some more coffee While finding these vulnerabilities is great, my goal is to figure out how these random photos ended up on the picture frame. I still didn't prove or disprove my theory that the built-in admin account sent the photos to the frame. The problem was lack of information. I had a good understanding of how the mobile application worked, and interacted with the back-end api, but little to no information on how the picture frame itself interacted with the back-end api. It was time to see if I could capture web traffic from the digital picture frame itself.

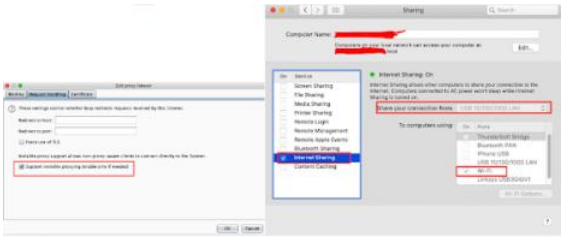
Unfortunately, there was a problem. I needed to somehow get the digital picture frame web traffic to proxy through BurpSuite. This is very easy with a web browser or a mobile application because both have a place to configure a proxy server setting. This proxy server setting allows all traffic from the web browser or mobile application to pass through BurpSuite which allows me to inspect and modify it. The digital picture frame does not have any way to configure a proxy server. The extent of the wifi configuration on the digital picture frame is selecting a wifi network and providing a passphrase to connect.

providing a passphrase to connect.

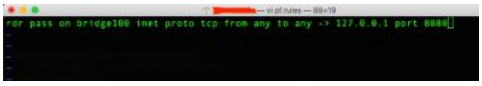


I did some research online because I've watched and attended countless IoT hacking talks, and this is a problem that thousands of other people have already solved. I felt a little silly because none of these talks discussed how they intercepted traffic from the device they were researching. This made me feel like it was so easy it wasn't worth documenting. After some google-fu, I found that BurpSuite has an invisible proxy setting. This means that any network traffic destined for the IP address BurpSuite is bound to will pass through proxy tab (not just the standard 8080 port traffic). This sounded like exactly what I needed, well it sounded like 50% of exactly what I needed. The other 50% was figuring out a way to connect the picture frame to an interface that was bound to BurpSuite. After some more research and a very helpful blog post, my plan was to create a wireless access point with my Macbook pro, that would use packet filtering (pf) to direct all network traffic to the BurpSuite interface.

First I enable invisible proxy mode on BurpSuite. Then I used an ethernet adapter as my primary network connection and shared my Internet connection using a wireless access point called "iot-test".

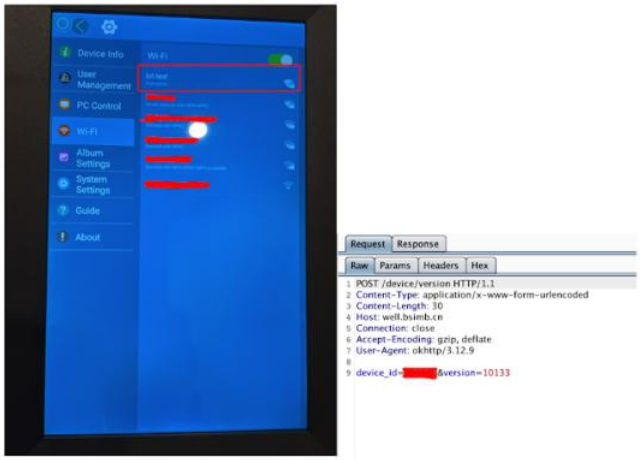


Then I created a pf.rules file to point all traffic from my bridged adapter to the interface BurpSuite (127.0.0.1:8080) and loaded the rule file.



Lastly, I joined the digital picture frame to the "iot-test" wireless access point I

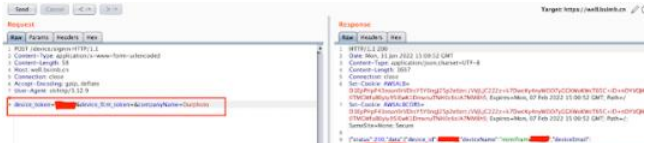
created and all the web traffic from the frame started passing through the proxy tab. Booyah!



More broken access control issues and clear-text passwords exposure Now that all the outbound web traffic from the picture frame was passing through my BurpSuite proxy, I started walking the different areas of the frames user interface and see what requests it would generate. One of the very first requests captured was to the end-point /device/signin. The device_token parameter was passed to this end-point. The device_token parameter is the unique identifier for each digital picture frame and it what is used to bind the Ourphoto mobile app to a frame. With the previous Insecure Direct Object Reference vulnerabilities I found I could scrape every single device_token for all of the digital picture frames in the database by incrementing or decrementing this sequential id number. This meant if I swap out the device_token id on the /device/signin end-point I could also see all of the device information for every picture frame in one place. The /device/signin end-point also made no attempt to implement the user_token parameter, so the end-point is completed unauthenicated. This end-point returned device_id, deviceName, deviceEmail, all user accounts bound to the frame (and their user_id's), and two other surprising pieces of data.

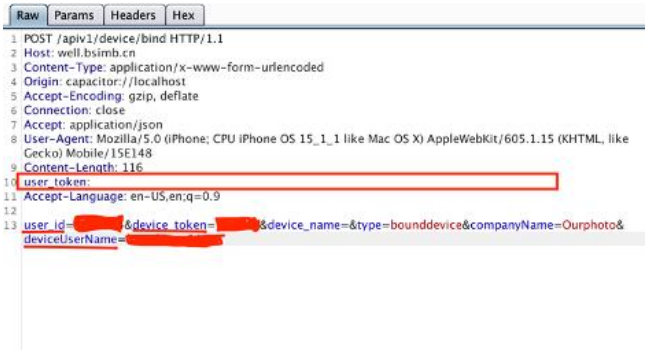
One of the additional sets of data appeared to be a username and password for video calling feature (deviceVideoCallUsername and deviceVideoCallPassword). I thought this was interesting, my digital picture frame was a bare bones model without a lot of features. I looked online at a few other more expensive models, and they included a camera and the ability to place video calls through the mobile application. I was confident if I had one of these more expensive models, I could likely place random video calls to any of the picture frames in the database.

The second set of data was the username and password for a MQ Telemetry Transport service. MQ Telemetry Transport (MQTT) is a protocol that many IoT devices use for their messaging broker. I am not sure of the implications of this account being exposed, but I had a gut feeling it wasn't a good thing.

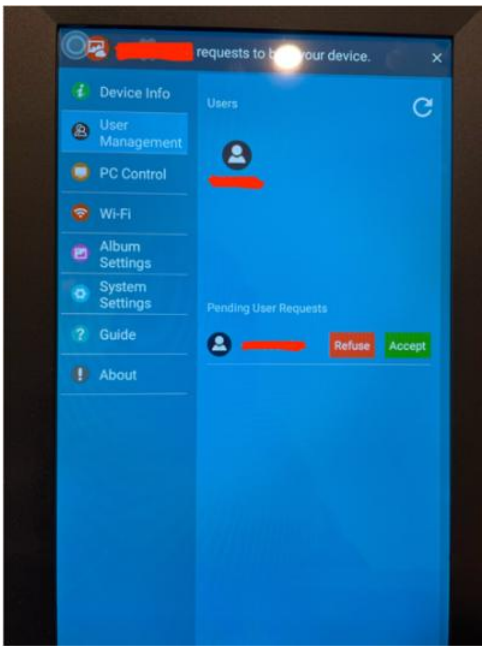


Getting side-tracked After finding the vulnerabilities and still not having an answer to how the random photos ended up on the frame, I got side-tracked. There clearly was a complete lack of access control on the Ourphoto mobile application and the digital picture frame, and where there was access control, it wasn't implemented properly. I started to ask the question, "Could I bind the Ourphoto application to my digital picture frame (or any users digital picture frame) without the end-user accepting the request on the touch screen?". If the answer to that was yes, then I could send photos and videos to any end-users picture frame without their consent. That would certainly drive the impact of these vulnerabilities up, but mostly it was for the lulz.

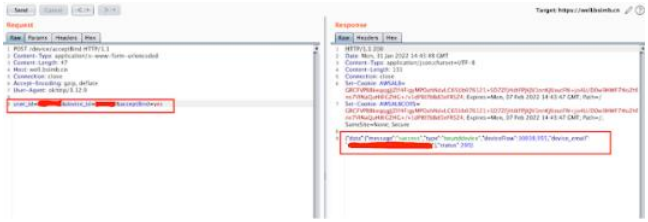
So from the mobile application, I went through the process of binding to the digital picture frame. The mobile application makes a request to /apiv1/device/bind end-point. The user_token header is present, but we know we can bypass it by removing the value. The process of binding only requires three pieces of information, the user_id, device_token, and deviceUserName. All three of these pieces of information we can get from the /apiv1/user/user_device and /device/signin end-points.



So, binding my user account through the Ourphoto mobile application to someone else's picture frame is no problem. No let's look at accepting that bind request on the picture frame without having physical access to it. When the request comes it to the picture frame it looks like the screenshot below.



Once the user clicks on the “Accept” button, the picture frame issues a web request to the /device/AcceptBind endpoint. This end-point also does not require authentication, and only requires the user_id and device_id for information. We have both pieces of information needed to accept our own bind request.



So, I ran through each step of sending a request to each end-point and I was successfully able to bind my user account through the Ourphoto mobile application to my digital picture frame without ever having to hit “Accept” on the screen. I’d also like to note I was also able to unbind or remove my account from the digital picture frame without having to click “Remove” from the “User Management” area. So technically, I could bind to someone else's picture frame, upload a photo, and remove myself from the frame without anyone knowing I did so.

Another way to send photos to other user's frames Since I was already relishing in the pageantry of being a threat-actor, I discovered a second attack vector. The digital picture frame provides the functionality to send photos via email. This is accomplished by the frame being assigned a unique email address, and the sender's email address needs to be added to the picture frame in a similar process as a Ourphoto mobile application user. The email also needs to use the frame's device_token in the subject line. This means to send a photo via email to the picture frame, the end-user needs to again click "Accept" on the picture frame to authorize the senders email account.

This got me thinking. What about email spoofing? The broken access control issues allow me to get both the frame unique email address, device_token and all the user emails that are authorized to send pictures via email. I configure postfix and whipped up a quick python script to spoof email.

```
import smtplib, ssl, email
from email import encoders
from email.mime.base import MIMEBase
from email.mime.text import MIMEText
from email.mime.multipart import MIMEMultipart

# start mail daemon on mac
# sudo postfix start

sender_email = "[redacted]@gmail.com"
receiver_email = "[redacted]@ourphoto.cn"

#MIMEMultipart
msg = MIMEMultipart("alternative")
msg["Subject"] = "[redacted]"
msg["From"] = sender_email
msg["To"] = receiver_email
filename = "/Users/[redacted]/Downloads/will.jpg"

#Message
html = """\
<html>
  <body>
    <p><b>Test</b>
    <br>
   test
</p>
  </body>
</html>
"""

part = MIMEText(html, "html")
msg.attach(part)

#Attach
with open(filename, "rb") as attachment:
    part = MIMEBase("application", "octet-stream")
    part.set_payload(attachment.read())

encoders.encode_base64(part)
```

I confirmed that if I spoofed an email address that was added to my frame, I could send pictures to the frame without any roadblocks. I ended up discovering that the subject line in the email which is supposed to contain the unique device_token is ignored. If I sent an email to the picture frame email address from an authorized email account with any subject line, the picture would make it to the frame. It would be easy for an attacker to scrape every authorized email and every unique frame email from the api, then send an email with the same picture to all the email addresses. I was thinking in terms of political messages, advertisements, or nefarious pictures.

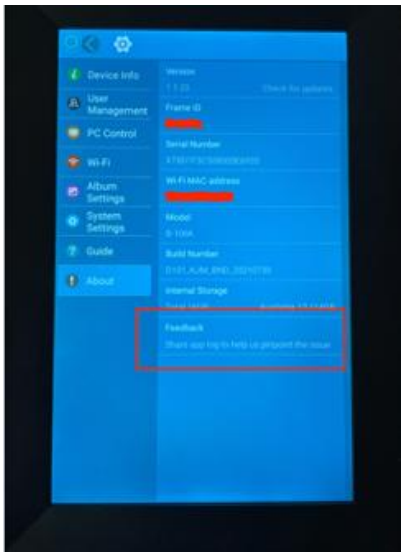
Exploiting the vulnerabilities I wrote a few python PoC's to exploit the discovered vulnerabilities. Below are a few video's showing the progression of my findings.

Adding a user

Adding a user and bypassing the user-acceptance dialog

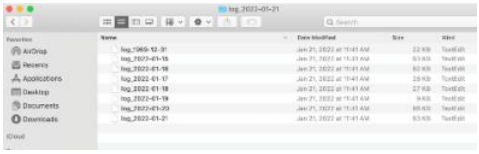
Adding a user, bypassing the user-acceptance dialog, and sending an unsolicited Richard pic to a frame

A breakthrough!!! So far I found decent amount of security issues with the Ourphoto mobile application and a few plausible ways another users photos could end up on Mimi's picture frame, I still didn't have a definitive answer of how it happened. Like most security researchers, I don't give up easily. It is a required trait to be successful in our security space. I started walking the user interface, and in the "About" section I found a "Feedback" option. The Feedback option allows you to "Share app log to help pinpoint the issue". I setup my BurpSuite interception proxy to capture the request and clicked the button.



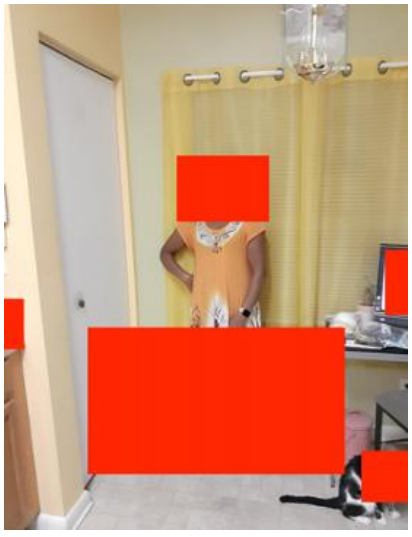
What I saw was a web request to upload a zip file with different log information in it from the picture frame. I copied the binary data from the web request and saved it off the file name being reference in the request (log_2022-01-21.zip). I unzipped the file and there I found all the application logs from the digital frame.

I unzipped the file and there I found all the application logs from the digital frame.



Found it!!! After reviewing all the log files and understanding their contents, I found what I was looking for. I made a list of the user_id numbers that belong to all the family members I had originally added to the picture frame. I then used grep and awk to look for user_id's that weren't in my list. That is when I got a hit on a user_id that was not in my list. The log entry provided the sender_id, sender_name, filename, url, sender_platform and some other information. On Sunday, January 16, 2022 at 7:50:17 PM a user sent the first picture to mimi's picture frame via email.

You can see in the log information that the "senders_platform" was email. This was interesting because I didn't authorize any of the family members email on the frame, just the mobile application. I looked up the user_id number against the api and found that it belonged to the built-in admin account I referenced in my original theory. The "url" parameter was interesting because it appeared to be a content delivery network address where the photo was uploaded before it hit the frame. Was this photo still available? Yup, it sure was. I found the random women whose picture showed up on mimi's picture frame.

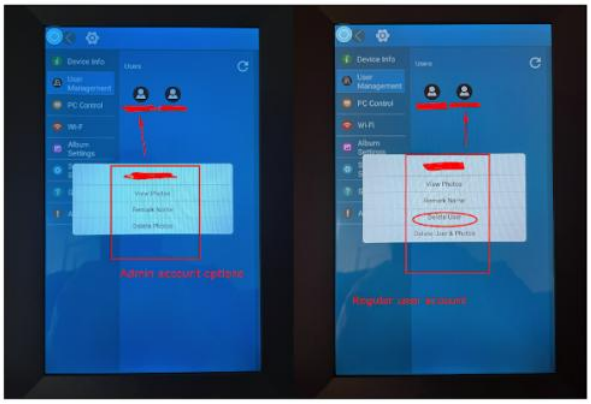


In total I found 6 pictures just as my mother-in-law had reported. I showed the pictures to my mother-in-law to confirm that they were indeed the person she saw. I

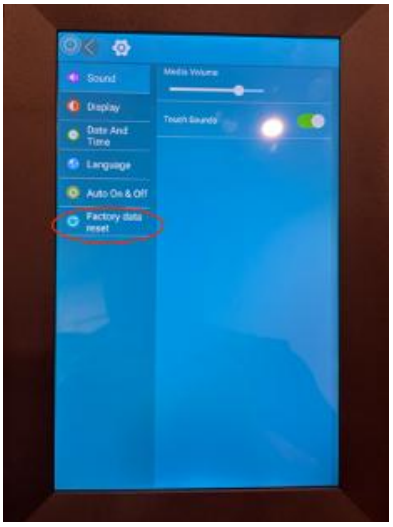
now know what the photo's look like and even how they ended up on Mimi's picture frame. I still didn't know why they ended up on Mimi's picture frame.

Solving the mystery So after a night of sleep, I started to think about the "why" this happened. My original theory that the photos were sent by the built-in admin account proved true. I started searching online support forms and technical documentation for the picture frame. I needed more information about the built-in admin account. What I found was puzzling, there was no reference to this built-in admin account.

I started clicking on the user accounts in the "User Management" section of the picture frame. I noticed the admin account doesn't give you an option to delete user, but any other user accounts did let you delete them. I thought it kind of made sense.



I decided to do a factory reset on the frame and to make sure nothing weird was going on with the built-in admin account. I used the option in the settings area of the user interface and waited for it to reboot. When it rebooted, I was greeted with a surprise. A setup menu for the frame I had never seen before. I thought that was interesting. I followed the new setup screen. Part of this setup was to provide a username which I had never seen before. When I completed the factory reset I found that the username I provided during setup was now the admin user account on the device. The admin account I now created was named differently than the built-in admin account I had been seeing before. This meant the admin account that came on the frame device was not setup by me, it was already on the device when I pulled it out of the box.



I now know why the photo's ended up on the frame. The digital picture frame we received from amazon was bought, used and returned. The person who returned the frame experienced the same thing I did in the user management screen, that there was no "delete" user option for the admin account. They deleted the other accounts, their photos, but did not do a factory reset. Then they returned the frame to amazon. We were sent the returned frame, and I added new user accounts to the frame while the old admin account still existed thinking it was a built-in admin account with a pre-configured name. The frame has "cloud" storage that comes with it, which is the referenced url to their content delivery network. When you connect the frame it checks the api if the user's photo on present on the frame, if they are not it pulls them down from cloud storage. At some point previous owner sent 6 photos via email, when the frame connected and it re-downloaded the users 6 photos.

There are more vulnerable devices I've reported the vulnerabilities to the Ourphoto App developer and after several months they released a fix for the mobile application. There is one last thing to note. These IoT digital picture frames are mass produced by same manufacturers in China and branded differently. This means the Ourphoto App is used on several different digital picture frames with different brand names. At the time of writing this post, I did a search on amazon for listings or reviews that references the Ourphoto App. I found at least 15 different brands of digital picture frames that use the Ourphoto App and were vulnerable to the discovered security issues. Below is a list.

1. KOT 10inch Wi-Fi Digital Picture Frame 1920x1080 IPS Touch Screen, Built in 16GB Memory, Share Photo and Video via OURPHOTO APP, Cloud, Email, Support Thumb USB Drive and SD Slot, Music Player (Black)
https://www.amazon.com/KOT-Digital-Picture-1920x1080-OURPHOTO/dp/B08T14Z96T

2. BIGASUO 14.1 Inch WiFi Digital Picture Frame, Touch Screen Smart Cloud Digital Photo Frame, Share Photos via OurPhoto App, Email & Cloud from Anywhere, 16GB Storage, Auto-Rotate, Wall- Mountable (Black)
https://www.amazon.com/Digital-Picture-Anywhere-Auto-Rotate-Mountable/dp/B08V15XR8D

3. ZTSWKJ WiFi Digital Picture Frame Touch Screen 10 inch -Digital Photo Frame 16GB Storage Auto-Rotate HD Smart Electronic Picture Frame family Share Photos & Videos via APP(OurPhoto),Email, Facebook & Twitter
https://www.amazon.com/Screen10-Unlimited-Auto-Rotate-Electronic-Instantly/dp/B08KDJ22KR?th=1

4. Dragon Touch
https://www.amazon.com/Dragon-Touch-Digital-Picture-Auto-Rotate/dp/B0815ZZ29C?th=1

5. BOIFUN Digital Picture Frame WiFi with 10.1“ IPS Touch Screen HD Display, Share Photo via App, Email, 16GB Storage, Wall-Mountable/Auto-Rotate/Support 1080P Video/USB/SD Card

https://www.amazon.com/Digital-Picture-Display-Wall-Mountable-Auto-Rotate/dp/B08CH5YXZ7

6. YEEHAO Digital Picture Frame WiFi Digital Photo Frame YEEHAO 1920x1080 Touch Screen, Support Thumb USB Drive and SD Slot, Music Player, Share Photos and Videos via APP, Cloud, Email(8 inch)
https://www.amazon.com/Digital-Picture-YEEHAO-1920x1080-Support/dp/B08M9FZTSP

7. Kimire Digital Photo Frame WiFi Digital Picture Frame kimire 1920x1080 Touch Screen, Support Thumb USB Drive and SD Slot, Music Player, Alarm Clock, Share Photo and Video via APP, Cloud, Email(10inch Black)
https://www.amazon.com/Digital-Picture-kimire-1920x1080-Support/dp/B08NCB7MQ7?th=1

8. CAMKORY CAMKORY WiFi Digital Picture Frame with Full HD IPS Touch Screen Display - 8 Inch LED Picture Frame Support Auto Rotate 8GB Internal Memory, Digital Photo Display with APP, Alarm, USB Port - Black
https://www.amazon.com/CAMKORY-Digital-Picture-Screen-Display/dp/B086HM7F83

9. AiJoy 10 Inch Digital Picture Frame with WiFi, 16GB Photo Frame with Touch Screen, HD 1280x800 IPS, Smart Share Photo via Email, APP, Facebook, Twitter, Support 1080P Video/Music, USB, SD Card
https://www.amazon.com/AiJoy-Digital-Picture-1280x800-Facebook/dp/B089SHFPYX

10 .GRC GRC 10.1 Inch WiFi Digital Photo Frame with IPS Full HD Touch Screen, Send Photos and Videos from App (iOS Android) Anywhere Anytime, 16GB Internal Storage, Support SD Card
https://www.amazon.com/GRC-Digital-Android-Anywhere-Internal/dp/B07ZJGN9T4

11. Atatat Digital Picture Frame WiFi 10 inch with 1920x1080 IPS Touch Screen, Share Photos & Videos Instantly via APP Email, Auto-Rotate, Wall-Mountable, Portrait and Landscape
https://www.amazon.com/Atatat-1920x1080-Instantly-Auto-Rotate-Wall-Mountable/dp/B08GXXK253

12. KEKEYANG Android Video Call Electronic Photo Album Display Set- up Home Touch WiFi Hd Digital Photo Frame Digital Photo Album (Color : Brown+16gtf Card)
https://www.amazon.com/KEKEYANG-Android-Electronic-Display-Digital/dp/B09ND59VGL

13. MRQ 10 inch WIFI Digital Photo Frame with 1280x800 IPS Touch Screen 16GB

Storage No Monthly Fee, Digital Photo Frame Share Photos via App, Email, Display Photos, Music, Video
https://www.amazon.com/MRQ-Digital-Cellphone-Internal-Included/dp/B07X1MWSM6

14. WXGA 10.1 Inch Full-View Electronic Photo Album, WiFi Cloud Digital Photo Frame, Sharing Photos Via APP, Smart HD Advertising Player, with 16G Memory
https://www.amazon.com/Full-View-Electronic-Digital-Sharing-Advertising/dp/B08DCHPHRL

15. HAODGUO 10.1 Inch Cloud Photo Frame, Smart Android System Video Call Electronic Photo Album, IPS High Definition Digital Photo Frame The Best Gift for Parents and Children
https://www.amazon.com/HAODGUO-Android-Electronic-Definition-Children/dp/B09GVXRHHB

Conclusion I figured out how the random women's photos ended up on the mimi's digital picture frame and found some security vulnerabilities along the way. The following CVE's were assigned for the vulnerabilities I discovered;

CVE-2022-24187
The user_id and device_id on the Ourphoto App version 1.4.1 /device/* end-points both suffer from insecure direct object reference vulnerabilities. Other end-users user_id and device_id values can be enumerated by incrementing or decrementing id numbers. The impact of this vulnerability allows an attacker to discover sensitive information such as end-user email addresses, and their unique frame_token value of all other Ourphoto App end-users.

CVE-2022-24188
The /device/signin end-point for the Ourphoto App version 1.4.1 discloses clear-text password information for functionality within the picture frame devices. The deviceVideoCallPassword and mqttPassword are returned in clear-text. The lack of sessions management and presence of insecure direct object references allows to return password information for other end-users devices. Many of the picture frame devices offer video calling, and it is likely this information can be used to abuse that functionality.

CVE-2022-24189
The user_token authorization header on the Ourphoto App version 1.4.1 /apiv1/* end-points is not implemented properly. Removing the value causes all requests to succeed, bypassing authorization and session management. The impact of this vulnerability allows an attacker POST api calls with other users unique identifiers and enumerate information of all other end-users.

CVE-2022-24190
The /device/acceptBind end-point for Ourphoto App version 1.4.1 does not require authentication or authorization. The user_token header is not implemented or present on this end-point. An attacker can send a request to bind their account to any

users picture frame, then send a POST request to accept their own bind request, without the end-users approval or interaction.

SHARE

Comments

To leave a comment, click the button below to sign in with Google.

Popular posts from this blog

Cracking password hashes on the cheap: How to rent online GPU resources for Hashcat

November 14, 2020

Summary In this post I will talk about how to make password cracking accessible to people who don't want to make the investment in video cards and GPU hardware resources. This post will mostly be a tutorial, geared toward people who are in a pinch and need to quickly standup a pc ...

SHARE 1 COMMENT READ MORE

Bypassing Windows Defender Antivirus in Windows Server 2016/2019

July 06, 2020

Summary In this post I will discuss a Windows Defender Antivirus bypass I discovered and reported to Microsoft on May 26th 2020. The bypass affects the current versions of Windows Defender deployed with Windows Server 2016/2019, where the Web Server role is installed. This post focu ...

SHARE 4 COMMENTS READ MORE

→

Follow

Author

1oopho1e - Nick M

.

VISIT PROFILE

Archive

Disclaimer

The information shared in this blog is for educational purposes only. This blog was created to help security researchers, bug bounty participants, and students learn about offensive security. I am not responsible for any illegal activity readers choose to engage in with the information provided. All videos and media content has been created on computer systems or applications owned by the author. All vulnerability information disclosed was done so with the permission of the vendors, or after a 90 day responsible disclosure period.