



# Why does my security camera scream like a Banshee?

Signal analysis and RE of a proprietary  
audio-data encoding protocol

# About Me

Rion Carter  
www: 7thzero.com

## Software Engineer

Love me some  
codes!

## Food

Recipe hacking with  
delicious results!

## Hacker Mindset

Move fast and break  
things... then fix  
them!

## Opinionated

All opinions are my own  
and may not reflect those  
of past/present employers

# TABLE OF CONTENTS

01

## Introduction

What are we doing here and why this wireless security camera?

02

## Signal Analysis

Capture, visualization and patterns

03

## App Analysis

Reversing the APK and digging deeper with Ghidra

04

## Hacking the Signal

Replicate the signal

05

## Demo

Master the Signal



01

## What Are We Doing Here?

And why are we talking about  
wireless security cameras?



**I** Original Goal: Use an inexpensive wireless camera to monitor the garden

# Inexpensive Camera



# Sounds easy... What's the catch?



Cloud 'App'  
Lock-in  
  
Vendor app  
requires a login



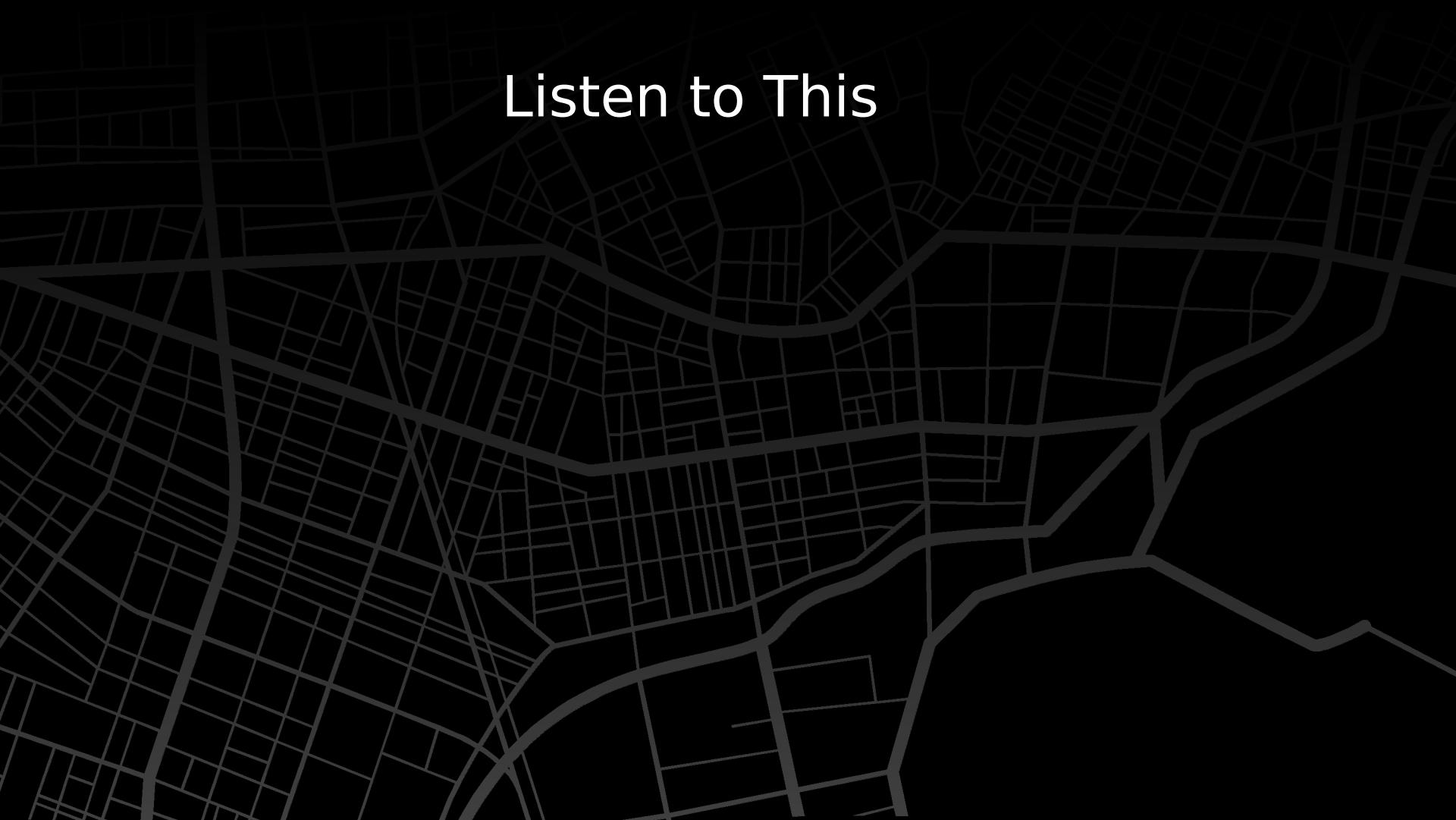
No docs  
  
Good luck finding  
reliable docs on a  
\$30 camera



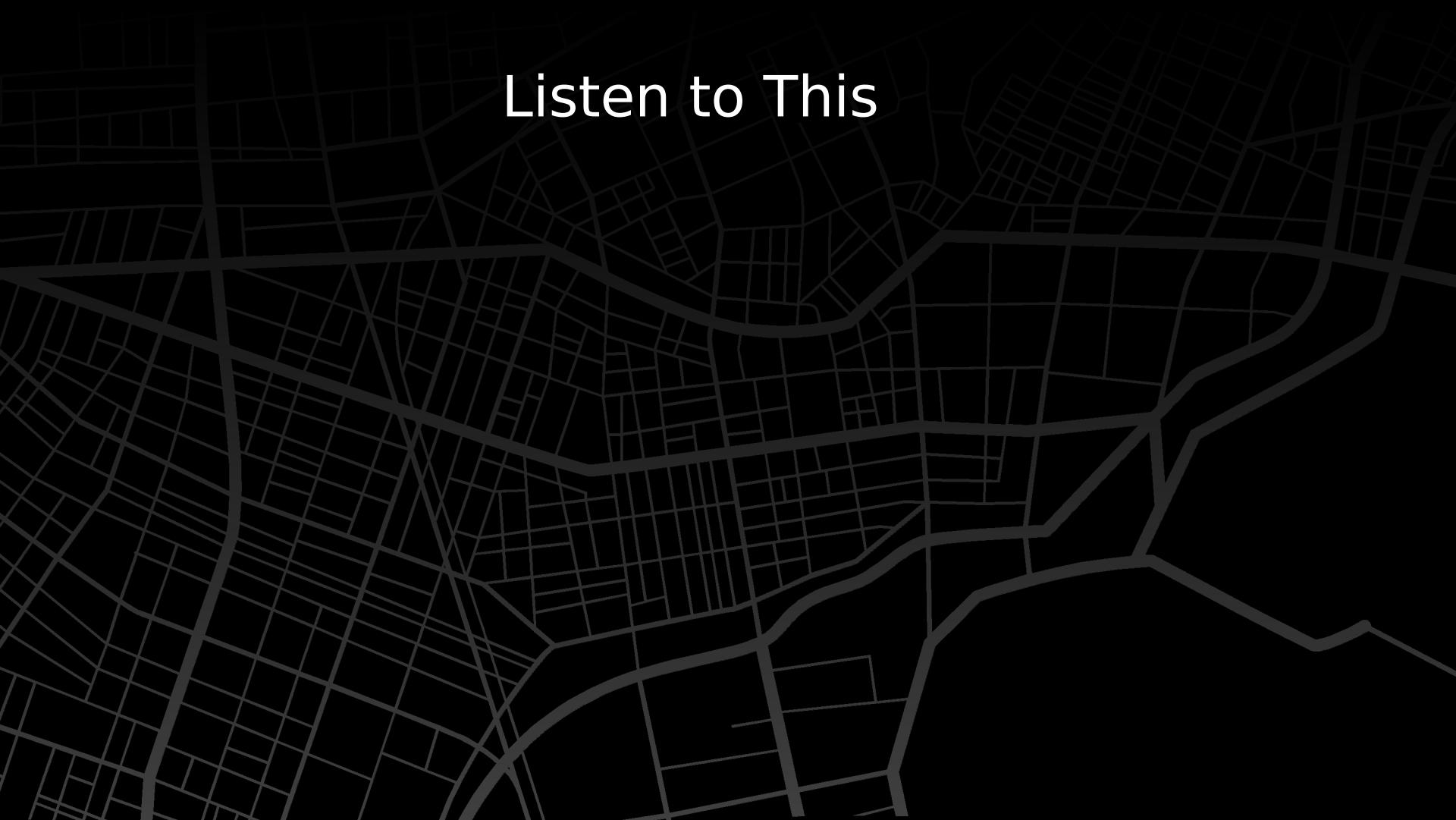
No self-setup  
  
No way to directly  
configure the camera  
(lacks BT/AD-HOC WIFI)



Bespoke Setup Protocol  
Using Soundwaves  
  
Which is what brings  
us here today ;)



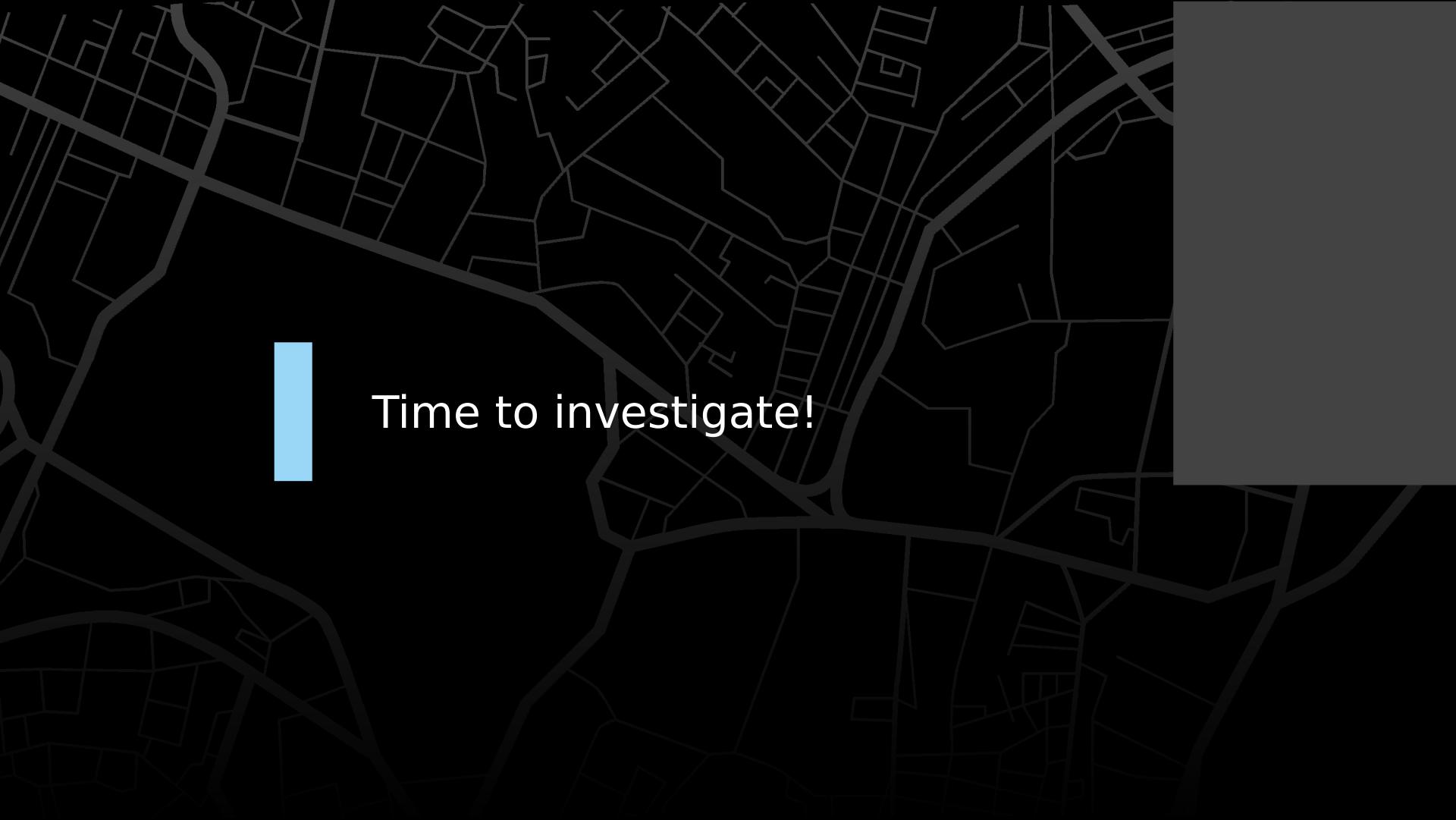
# Listen to This



Listen to This



New Goal: Figure out what is  
going on during camera setup



Time to investigate!



## Hardware

# Camera Inspection



USB

Unfortunately, this  
is not an avenue of  
investigation

The cable supplies  
power to the camera

# Camera Inspection



## BT/WIFI

On boot, the camera enters 'setup mode'

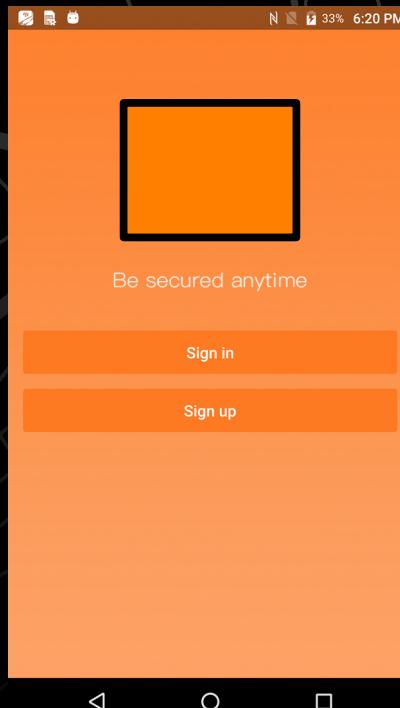
Setup mode requires the app for pairing

The camera **does not** advertise an Ad-Hoc network or show up as a bluetooth device



Software

# Vendor App

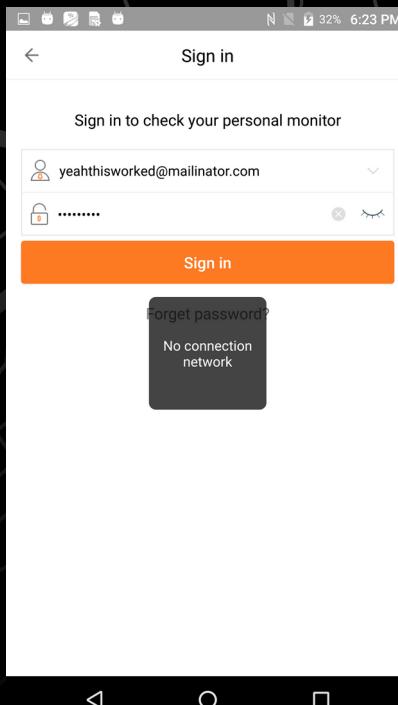


## Meet JAWA

JAWA is the app used to configure the 'Cloud Camera'

The App needs  
something...

Unfettered  
internet access

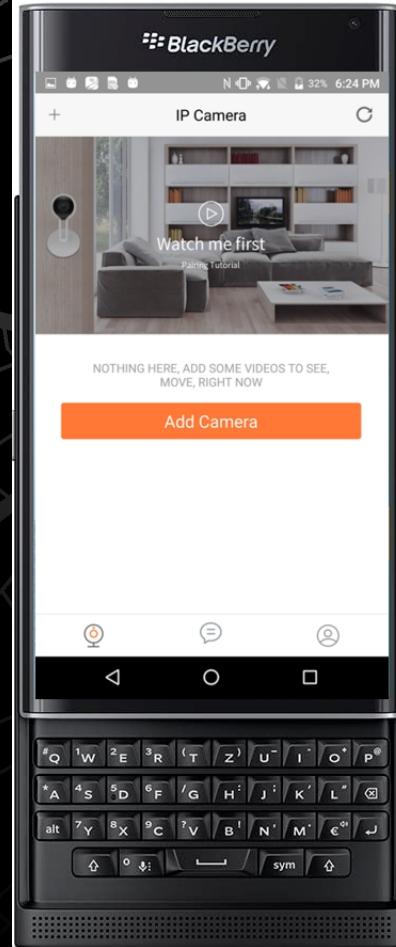


Vendor App

# Vendor App

Time for a  
'test' device

All vendor-app  
analysis performed  
on a 'vintage'  
BlackBerry Priv



# Not much online...

Searching for terms like 'Amiccom', 'Smart IP Camera', and 'audio pairing wifi camera' didn't turn up a whole lot. Expanding the search found additional cameras that may use a similar pairing technique, though...

- Imou Cue 2, Ranger 2, LOOC, Bullet Lite, Ranger Pro

Interestingly enough, I found examples of cheap cameras that use a less-flaky approach (QR Code Scanning) in place of audio pairing:

- BESDER 1080P Cloud Storage Wireless PTZ IP Camera
- Swann Indoor/Outdoor Wireless 1080p WiFi Security Camera

02

# Signal Analysis

It sounds... Intriguing?

# What should we try?

## Capture & Visualize

Capture the signal,  
use tools to  
visualize the signal

## Reptition

Are there any parts  
of the signal that  
repeat?

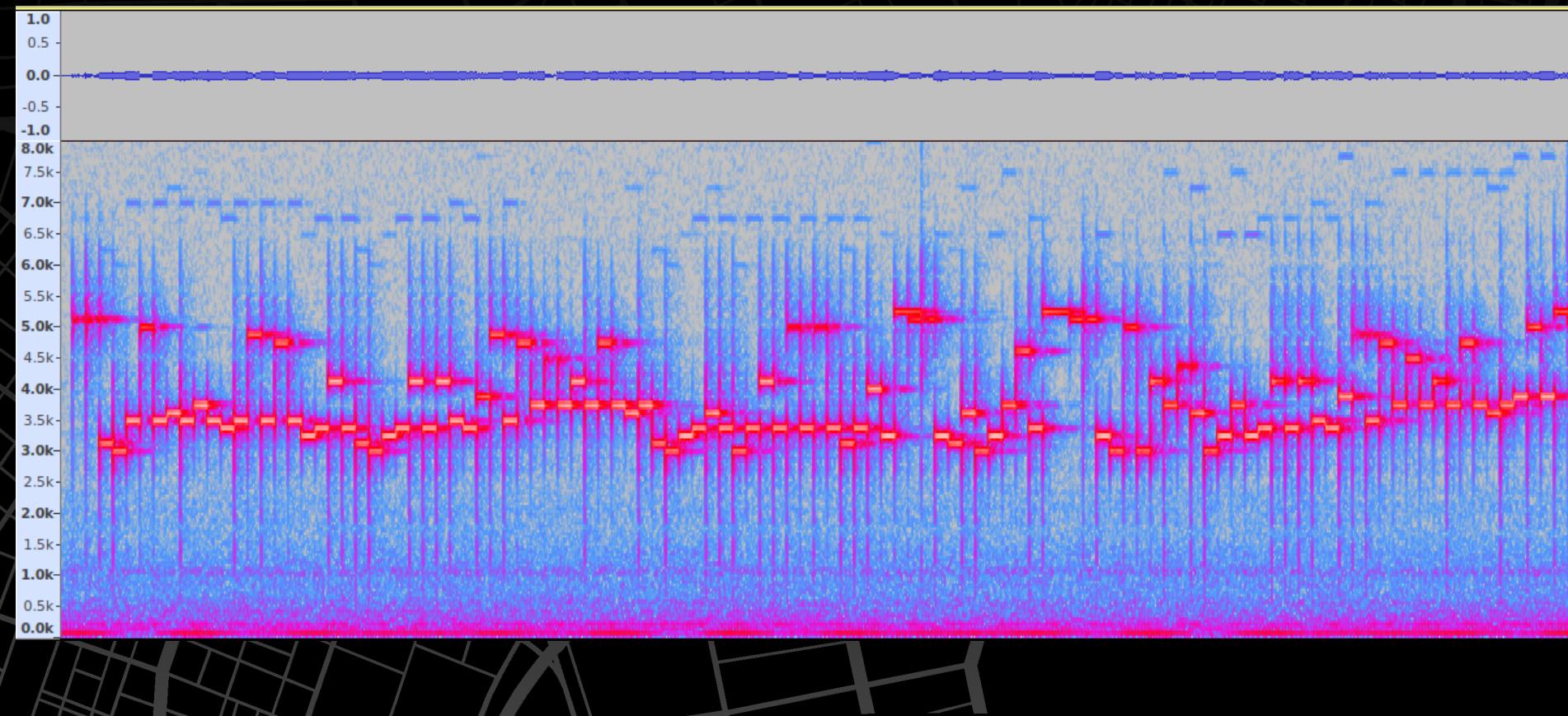
## Replay & Variation

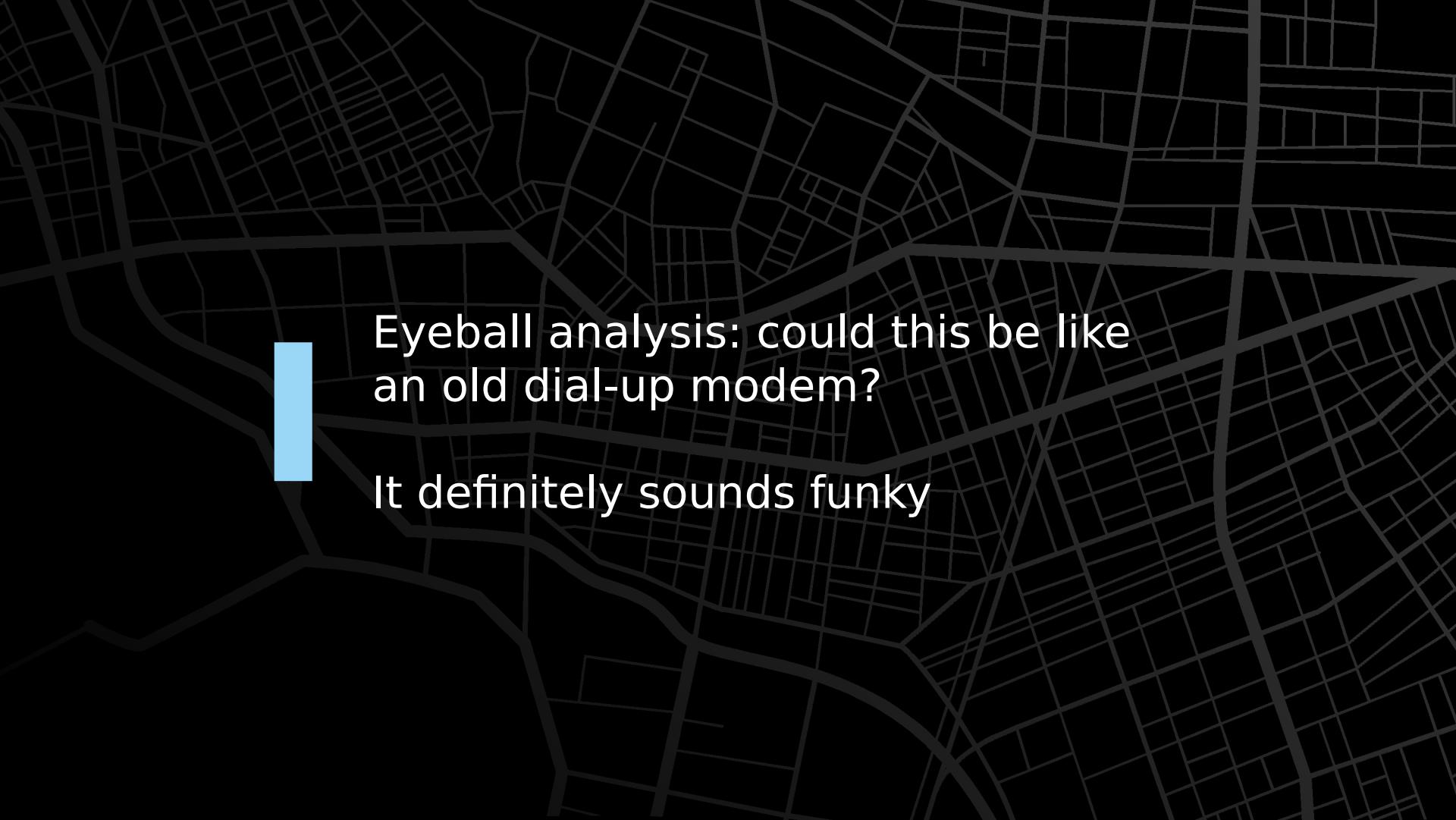
What differences stand  
out? Any outliers?  
Patterns where variated  
output is observed?

## Fuzzing / Simulation

Take control of the  
input and examine  
the output

# Capture viewed in Audacity





Eyeball analysis: could this be like  
an old dial-up modem?

|

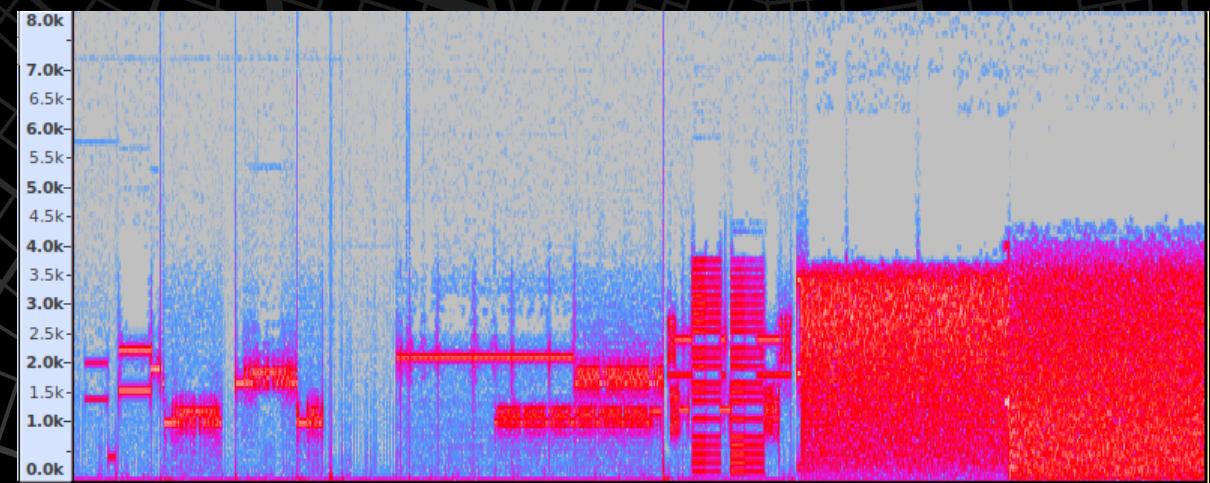
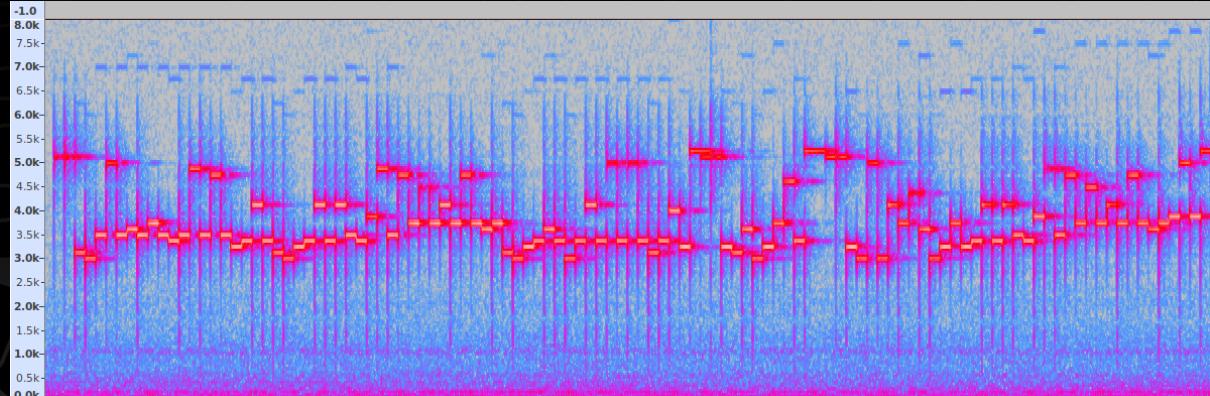
It definitely sounds funky

Confirmed:  
Not a Modem

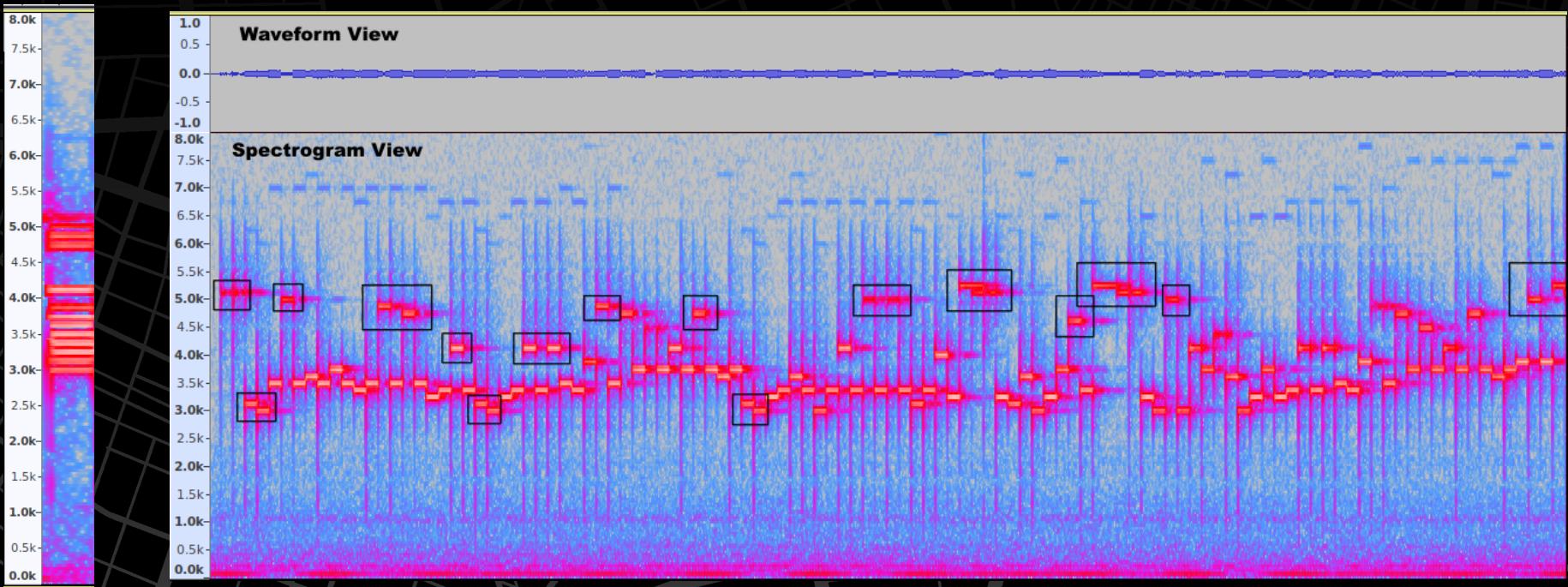
Camera Comm Signal

56k Modem

Spectrographs are  
substantially different



# Eyeball analysis



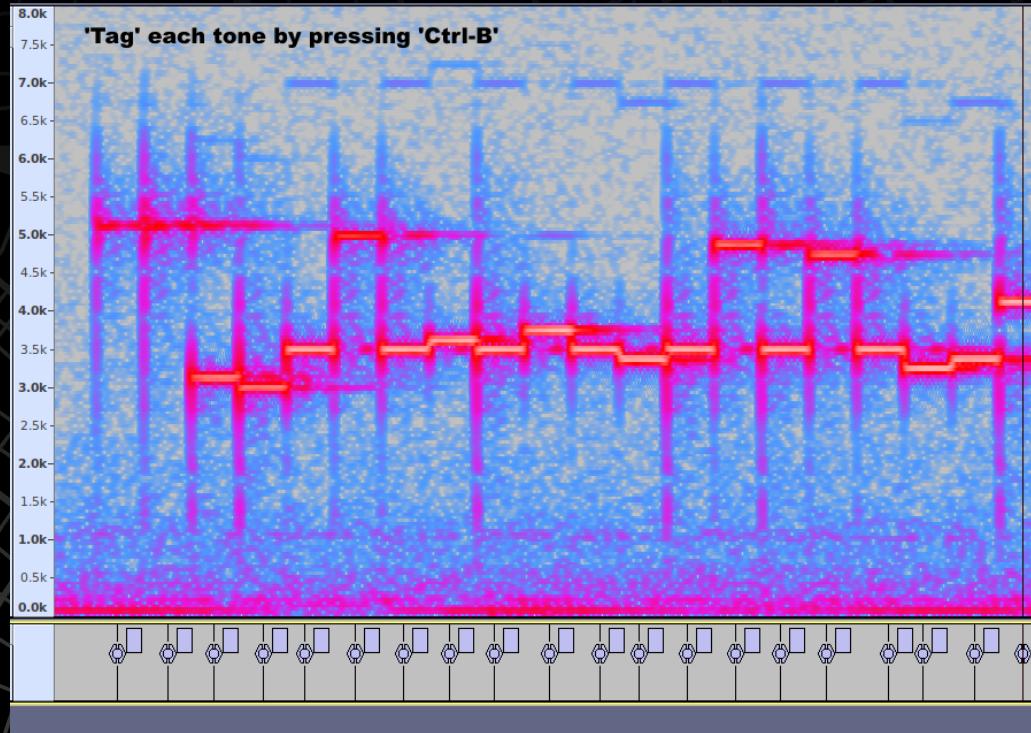
- Collapsed-Spectrograph view shows output in 3000Hz → 5000Hz range
- Quick eyeballing of the graph shows a few areas of interest



Can we get more precise?

Pretty pictures can only take us so far

# Manual Mode



Use Labels in Audacity:

- Position the cursor over each frequency time-slice
- Press **Ctrl-B** to add the label
- This will cause Audacity to calculate the frequency of each tone

# Manual Mode

Press F2 or double click to edit cell contents.

Track	Label	Start Time	End Time	Low Frequency	High Frequency
1 - Label Track		00 h 00 m 21.008 s	00 h 00 m 21.008 s	5,101.89 Hz	5,101.89 Hz
1 - Label Track		00 h 00 m 21.072 s	00 h 00 m 21.072 s	5,041.51 Hz	5,041.51 Hz
1 - Label Track		00 h 00 m 21.130 s	00 h 00 m 21.130 s	3,335.85 Hz	3,335.85 Hz
1 - Label Track		00 h 00 m 21.193 s	00 h 00 m 21.193 s	2,943.40 Hz	2,943.40 Hz
1 - Label Track		00 h 00 m 21.245 s	00 h 00 m 21.245 s	3,532.08 Hz	3,532.08 Hz
1 - Label Track		00 h 00 m 21.309 s	00 h 00 m 21.309 s	4,709.43 Hz	4,709.43 Hz
1 - Label Track		00 h 00 m 21.370 s	00 h 00 m 21.370 s	3,683.02 Hz	3,683.02 Hz
1 - Label Track		00 h 00 m 21.428 s	00 h 00 m 21.428 s	3,652.83 Hz	3,652.83 Hz
1 - Label Track		00 h 00 m 21.484 s	00 h 00 m 21.484 s	3,501.89 Hz	3,501.89 Hz
1 - Label Track		00 h 00 m 21.554 s	00 h 00 m 21.554 s	3,743.40 Hz	3,743.40 Hz
1 - Label Track		00 h 00 m 21.618 s	00 h 00 m 21.618 s	3,501.89 Hz	3,501.89 Hz
1 - Label Track		00 h 00 m 21.667 s	00 h 00 m 21.667 s	3,350.94 Hz	3,350.94 Hz
1 - Label Track		00 h 00 m 21.728 s	00 h 00 m 21.728 s	3,516.98 Hz	3,516.98 Hz
1 - Label Track		00 h 00 m 21.789 s	00 h 00 m 21.789 s	4,664.15 Hz	4,664.15 Hz
1 - Label Track		00 h 00 m 21.846 s	00 h 00 m 21.846 s	3,909.43 Hz	3,909.43 Hz
1 - Label Track		00 h 00 m 21.904 s	00 h 00 m 21.904 s	4,649.06 Hz	4,649.06 Hz
1 - Label Track		00 h 00 m 21.982 s	00 h 00 m 21.982 s	3,728.30 Hz	3,728.30 Hz
1 - Label Track		00 h 00 m 22.026 s	00 h 00 m 22.026 s	3,245.28 Hz	3,245.28 Hz
1 - Label Track		00 h 00 m 22.091 s	00 h 00 m 22.091 s	3,381.13 Hz	3,381.13 Hz
1 - Label Track		00 h 00 m 22.152 s	00 h 00 m 22.152 s	4,150.94 Hz	4,150.94 Hz

Insert  
Delete  
Import...  
Export...

Cancel OK ?

View the **Labels** in Audacity:

- Click on **Edit** → **Labels** → **Edit Labels...**
- Audacity will show a list of labels
- This list of labels will include frequency information
- Press the **Export** button to get this in a text file for analysis

# Manual Mode

Zoomed-view of frequency detection

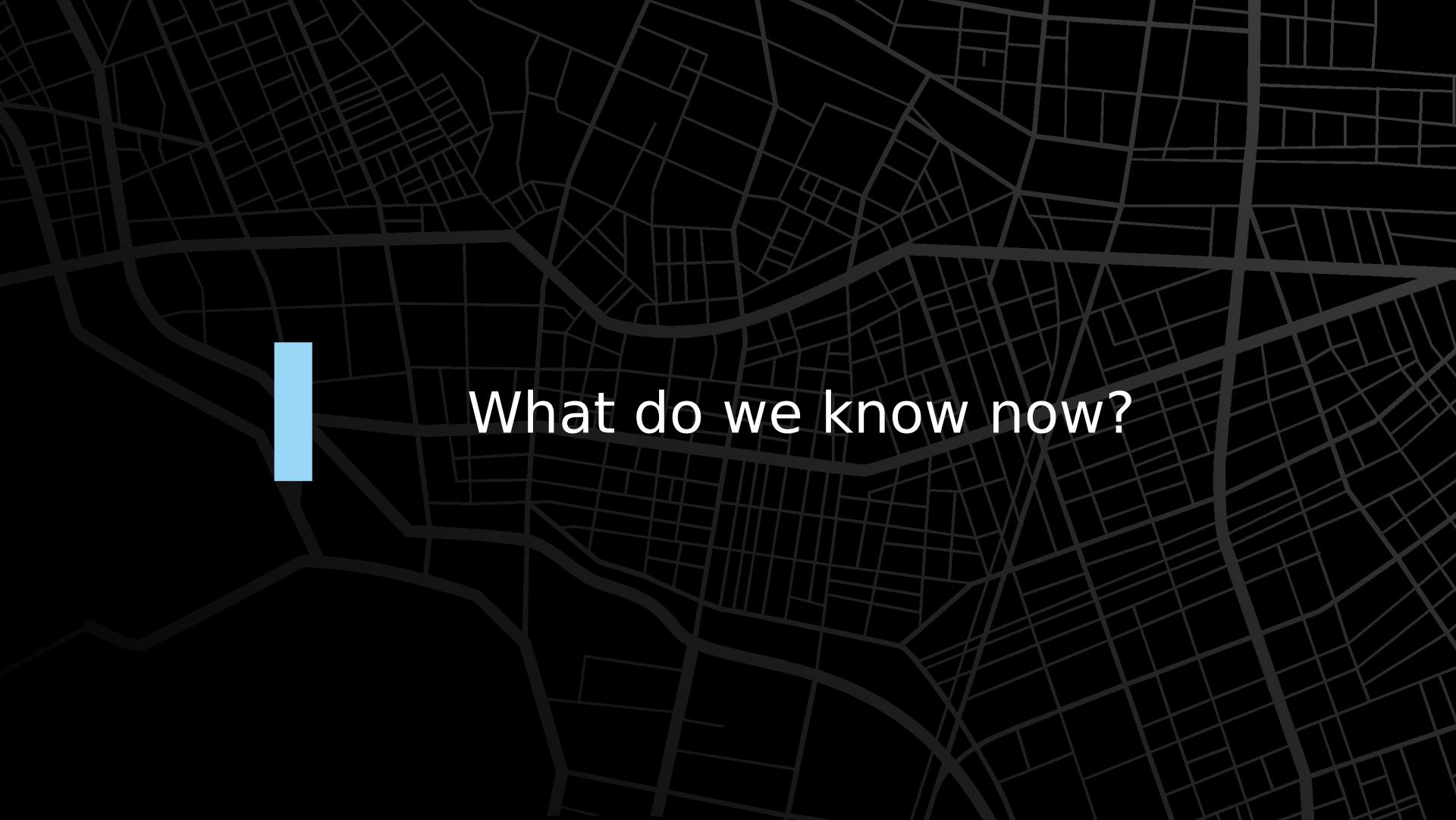
Low Frequency	High Frequency
5,101.89 Hz	5,101.89 Hz
5,041.51 Hz	5,041.51 Hz
3,335.85 Hz	3,335.85 Hz
2,943.40 Hz	2,943.40 Hz

[Insert](#)

[Delete](#)

[Import...](#)

[Export...](#)



What do we know now?

# What do we know now?



## Tone-encoded

Despite the audible-medium, the signal is digitized in coded tones



## Consistent Range

Tones are in the 3Khz to 5Khz range



## Control Signals...?

Outliers at the top and bottom of the spectrum warrant a closer look



## Repetition

The complete sequence repeats several times



## Not 'Binary'

The signal isn't two-toned or 'directly' sending 1's and 0's



## Not 56k Modem!

Spectral analysis does not match



Can we go further with  
Manual Mode?

# Can we go further manually?

Yes, with caveats:

- **Variability** of audio playback and capture gets old
  - You'll tire of this quickly
- It would be great to identify **exact** frequencies used
  - Audacity frequency analysis gets you 'close'
  - You'll find variation in the detected frequency depending on which tone you check
- **No API** is readily available to leverage this Audacity functionality
  - Also no readily available CLI option for us to script...

# What are the manual mode options?

Black-box signal reversing

- Brute-force reproduction of tones
- Attempt to match generated tones with the spectrograph
- Fuzz/generate permutations of the signal until you find a match

We have a better way...



Next Up: Reverse-engineer the  
App to understand the signal  
protocol

# 03

## App Analysis: Reversing the APK

We have the vendor app, so  
let's take a closer look



What are our options?

# What are our options?



## Execute and Log

Leverage Android debug log to picture the runtime state



## String Search

Look for strings or constants to aid in our understanding



## Decompile

Reverse compiled code



## Key Methods

Obfuscation may not strip out all meaning



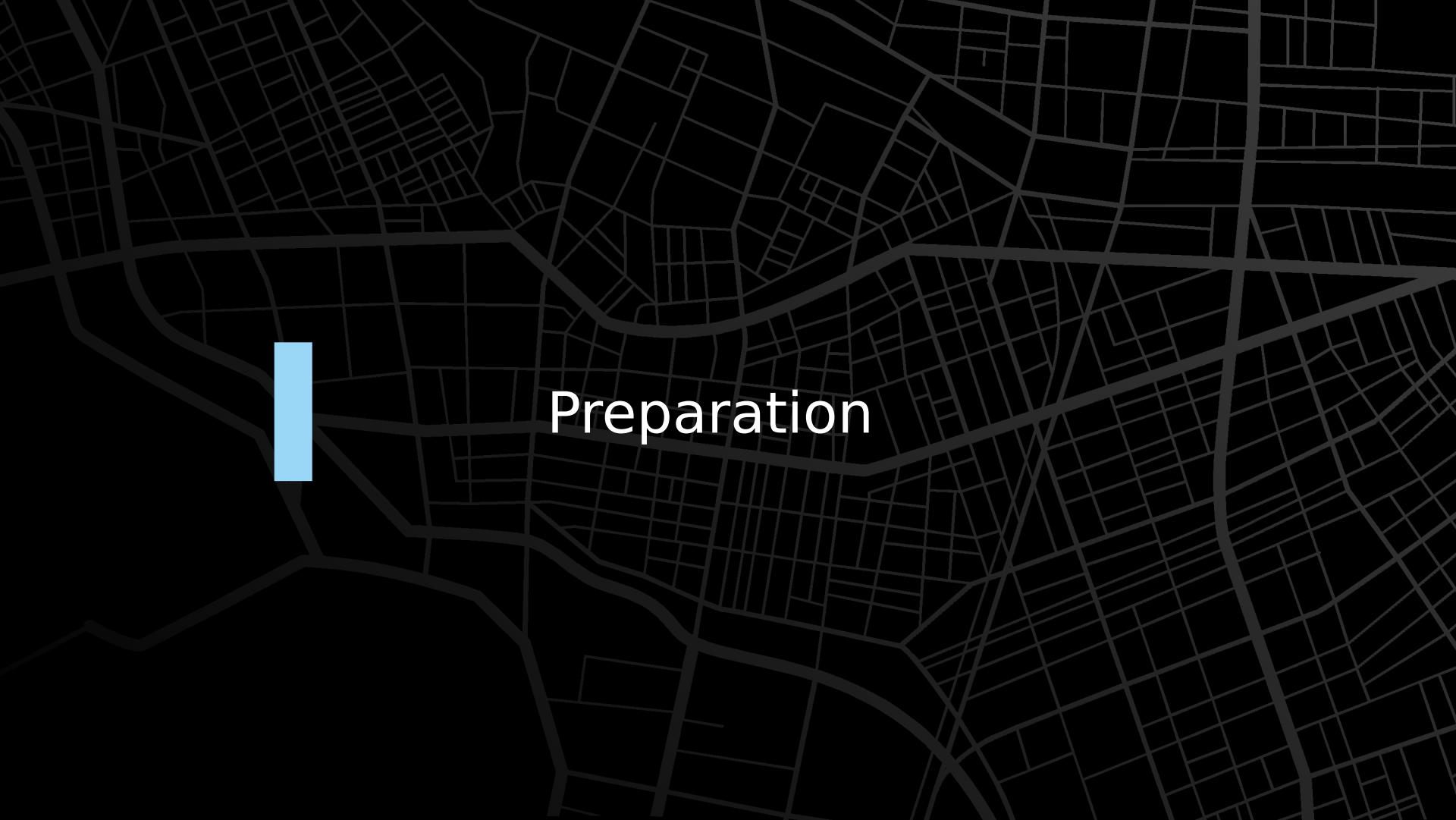
## De-Obfuscate

Attribute meaning to code that's lost its context



## High-Speed Fuzz

Leverage code to automate brute-force where necessary



Preparation

# Prepare Your Computer to Pull the APK

1. Enable **Developer Mode** on your Android test device
  - Go to 'Settings' and find the 'Software Information' for your device
  - Tap the **Build number** field 7 to 10 times
2. Allow **USB Debugging**
  - Go to 'Settings' and find 'Developer options' (which appears after step 1)
  - Flip the slider to enable/allow 'USB debugging' when USB is connected
3. Ensure you have **Android Studio** installed (includes the adb command)
4. Verify that **adb** is correctly installed and that USB debugging is properly enabled on your test device:
  - Plug in your device
  - Run this command:
    - `adb shell pm list packages | tail`

(You should see a list of packages)

# Extract the APK

To find and extract the camera app APK, follow these commands:

1. Open a terminal which can execute `adb`
2. Execute: `adb shell pm list packages | awk -F':' '{print $2}' | grep jawa`
3. With the package name from step 2, execute: `adb shell pm path com.jawa.cam | awk -F':' '{print $2}'`
4. Use output from step 3, execute: `adb pull /data/app/com.jawa.cam-1/base.apk`  
(Note: The path varies per-device, so use your output!)
5. Rename the APK to something usable:  
`mv base.apk com.jawa.cam.apk`
6. Sample Output:

```
rion@0x:~/Desktop$ adb shell pm list packages | awk -F':' '{print $2}' | grep jawa
com.jawa.cam
rion@0x:~/Desktop$ adb shell pm path com.jawa.cam | awk -F':' '{print $2}'
/data/app/com.jawa.cam-1/base.apk
rion@0x:~/Desktop$ adb pull /data/app/com.jawa.cam-1/base.apk
/data/app/com.jawa.cam-1/base.apk: 1 f.... 4.1 MB/s (44495777 bytes in 10.413s)
rion@0x:~/Desktop$ mv base.apk com.jawa.cam.apk
```

# Decompile the APK: JADX

For this analysis, I used JADX

1. Visit <https://github.com/skylot/jadx/releases>
2. Download the latest release
3. Extract the archive
4. `./jadx ~/tmp/apk/com.jawa.cam.apk -d ~/tmp/apk/jawa-decompiled`

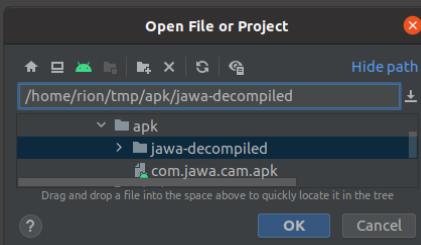
Example

```
rion@0x: ./jadx ~/tmp/apk/com.jawa.cam.apk -d ~/tmp/apk/jawa-decompiled
INFO  - loading ...
INFO  - processing ...
ERROR - finished with errors, count: 15
rion@0x: █
```

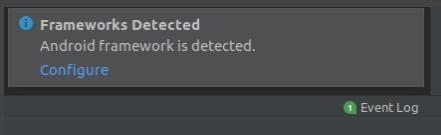
# Decompile the APK: Android Studio

Load the decompiled APK into an IDE

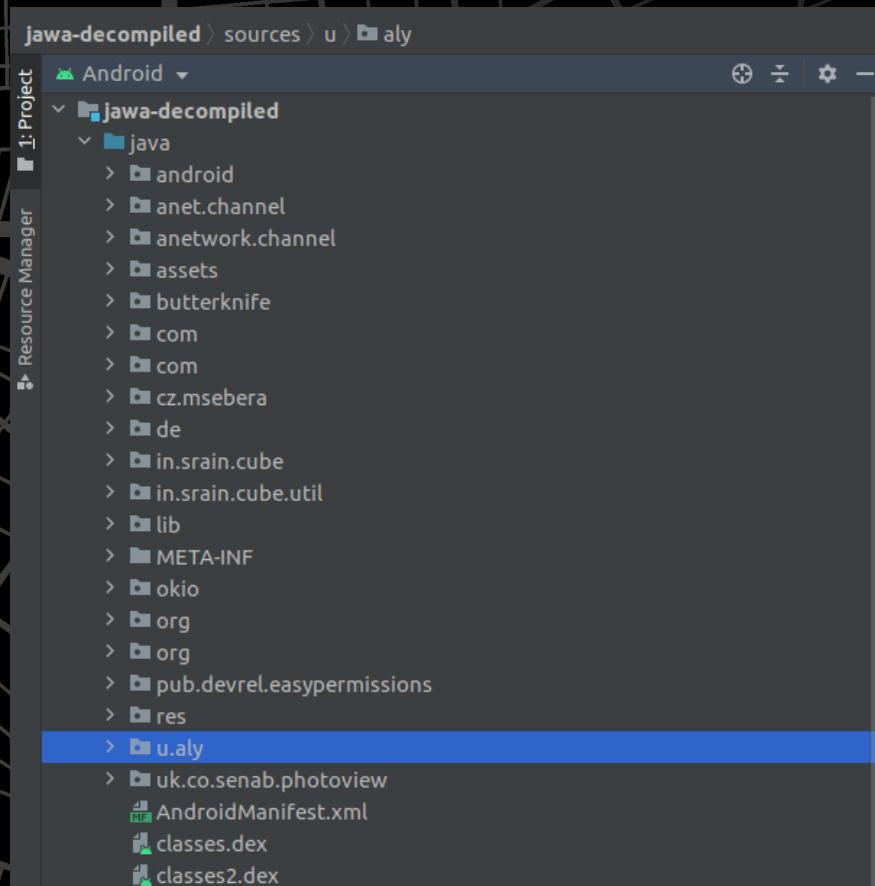
1. Ensure you have [Android Studio](#) installed
2. Select [File -> Open](#) and select your decompiled sources



3. Click the [Configure](#) button to setup the Android framework

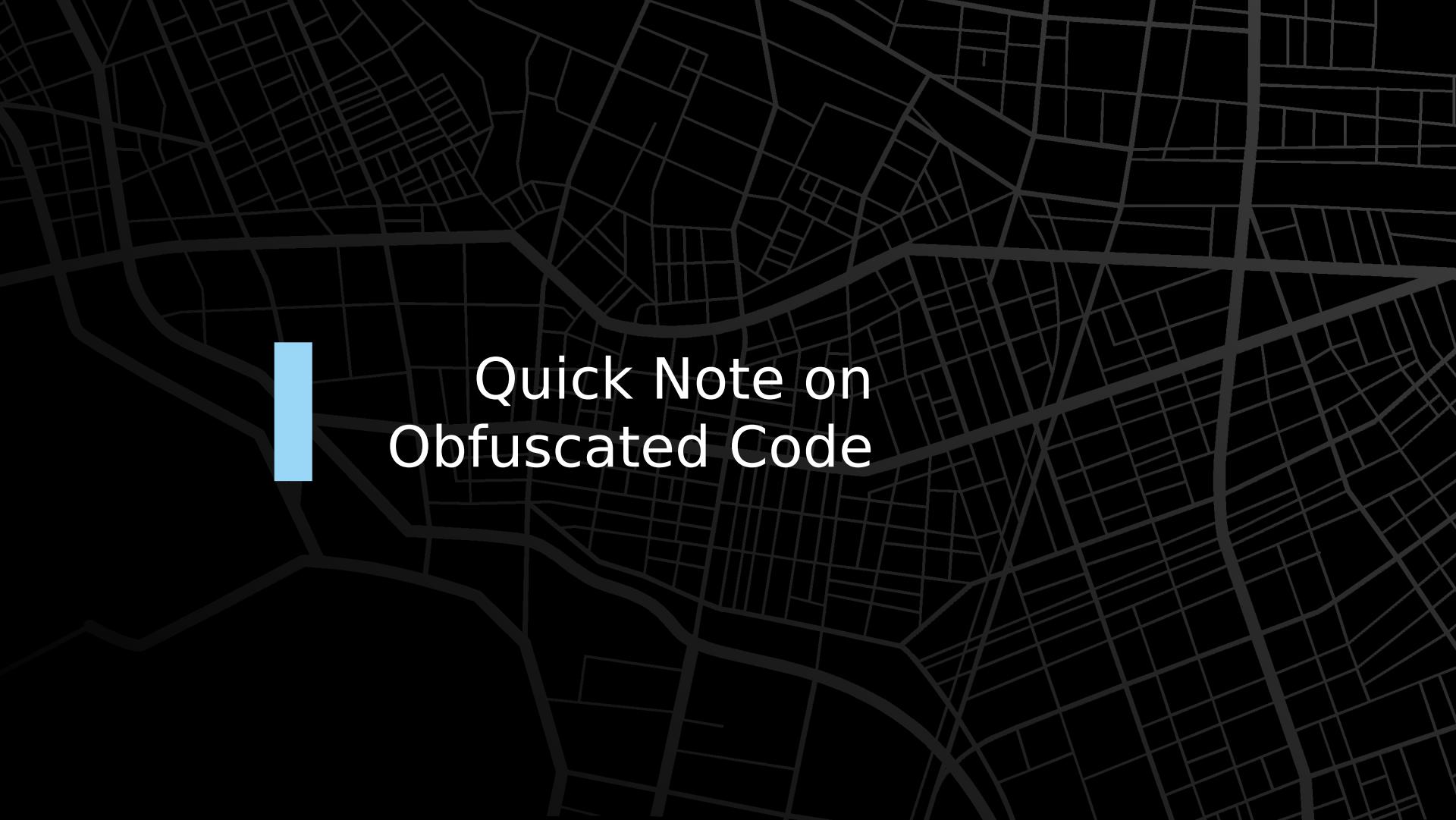


# Decompile the APK: Android Studio



Behold: the loaded project!

It holds the Secrets of the Grid



# Quick Note on Obfuscated Code

# A Note on Obfuscation

What is obfuscation?

- Software makers try to hide what their software does
- With higher level languages you get terse, ‘randomly’ generated identifiers
- Harder to obfuscate the use of system libraries than their own code



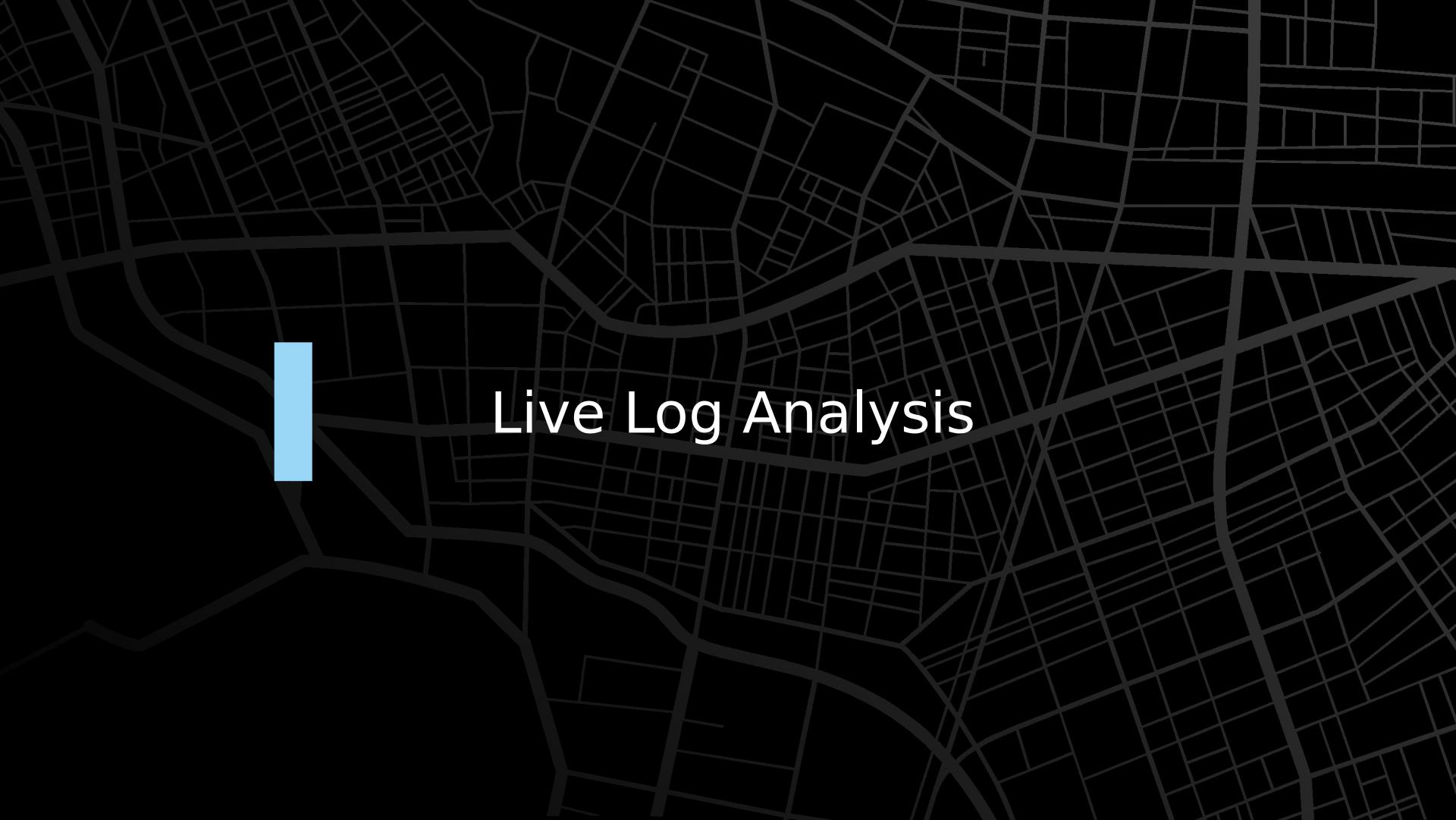
# A Note on Obfuscation

Why **Android Studio** for de-obfuscation?

- Easily re-factor throughout the project
- **Find Usages** and **Go To Declaration**
- Integrated LogCat search bar

... Still a frustratingly manual process

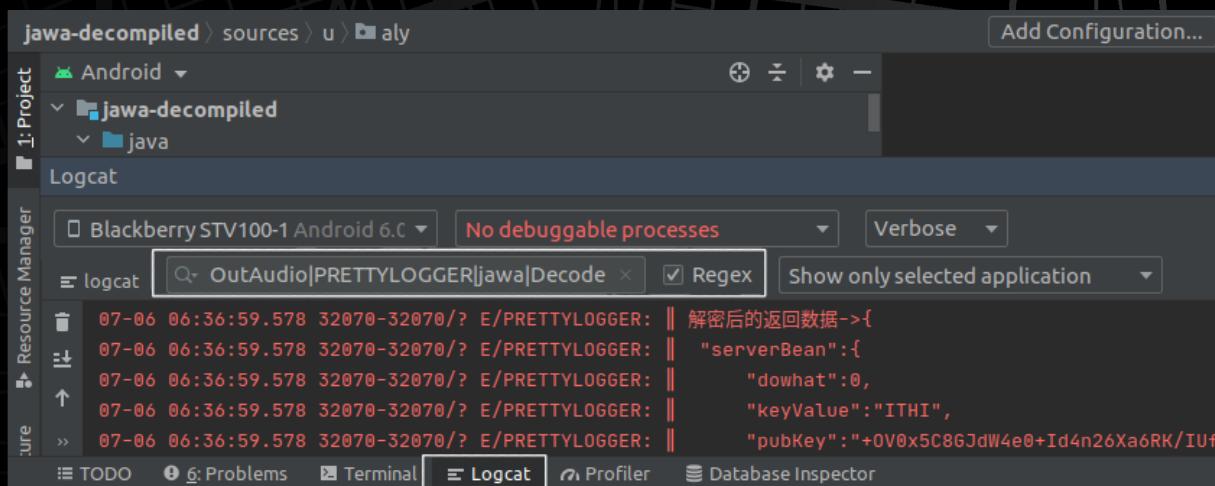




# Live Log Analysis

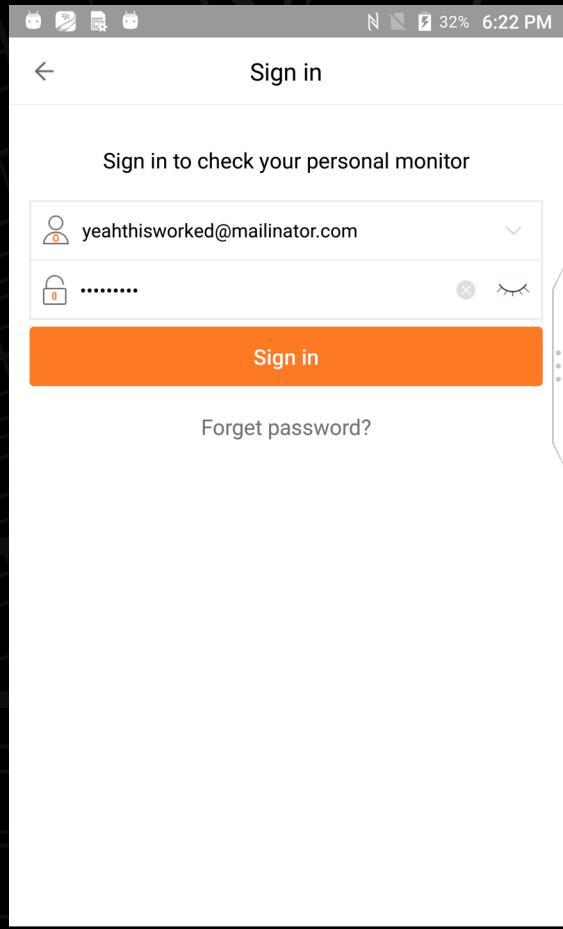
# Live Log Analysis Setup

- Plug in your Android **Test Device**
- Select the **LogCat** tab in the bottom window
- Enter this in the **Search** field and check the **Regex** box:
  - OutAudio|PRETTYLOGGER|jawa|Decode





# Logs on App Startup

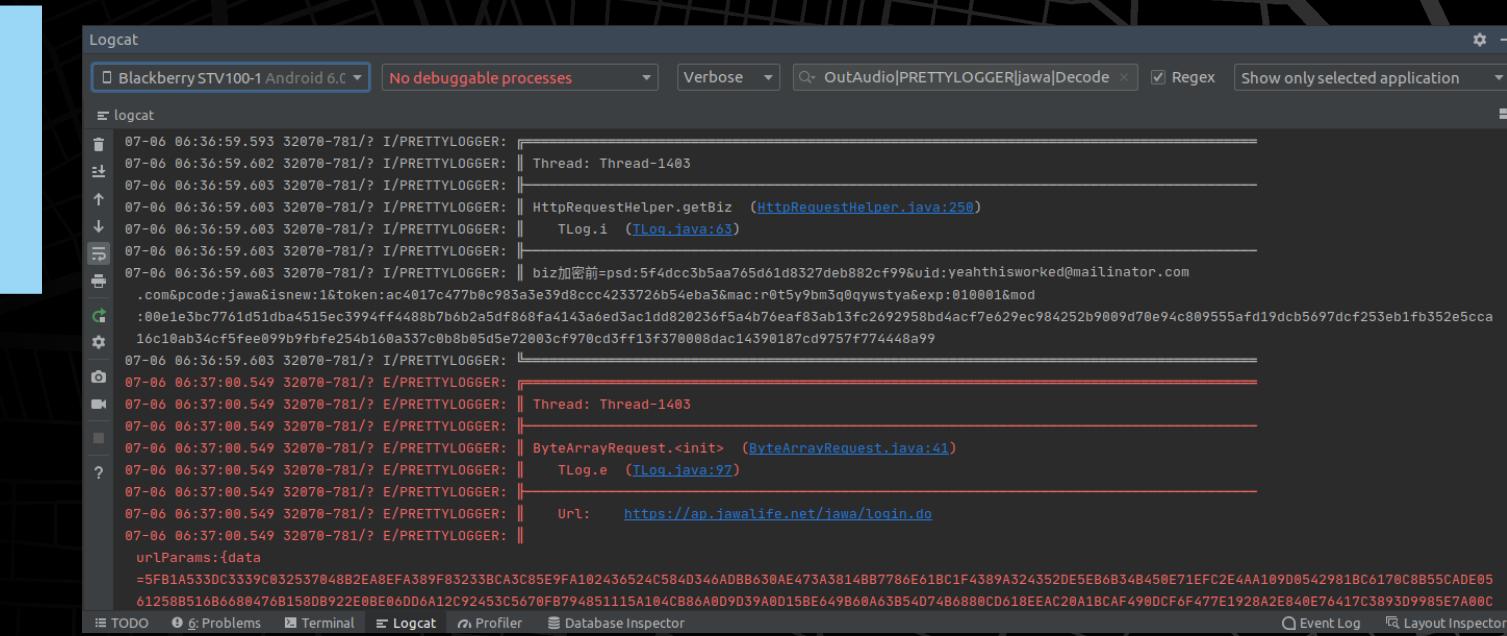


# Live Log Analysis

Login Screen

# Live Log Analysis: App Startup/Login

Hmm... looks interesting



The screenshot shows the Android Studio Logcat window with the following configuration:

- Device: BlackBerrySTV100-1 Android 6.0
- Filter: No debuggable processes
- Log Level: Verbose
- Search: OutAudio|PRETTYLOGGER|jawa|Decode
- Regex: checked
- Show only selected application: checked

The log output is as follows:

```
07-06 06:36:59.593 32070-781/? I/PrettyLogger: Thread: Thread-1403
07-06 06:36:59.602 32070-781/? I/PrettyLogger: HttpRequestHelper.getBiz (HttpRequestHelper.java:250)
07-06 06:36:59.603 32070-781/? I/PrettyLogger: TLog.i (TLog.java:63)
07-06 06:36:59.603 32070-781/? I/PrettyLogger: 
07-06 06:36:59.603 32070-781/? I/PrettyLogger: biz加密前=psd:5f4dcc3b5aa765d61d8327de0882cf99&uid:yeahthisworked@mailinator.com
07-06 06:36:59.603 32070-781/? I/PrettyLogger: .com&pcode:jawa&isnew:1&token:ac4017c477b0c983a3e39d8ccc4233726b54eba3&mac:r0t5y9bm3q0qywstya&exp:010001&mod
07-06 06:36:59.603 32070-781/? I/PrettyLogger: :00e1e3bc7761d51dba5151ec3994ff4488bf7b6b2a5df868fa414a6ed3ac1dd820236f5a4b70eaef83ab13fc2692958bd4acf7e629ec984252b9009d70e94c809555af
07-06 06:36:59.603 32070-781/? I/PrettyLogger: 16c10ab34cf5fee099b9fbfe254b160a337c0b08b05d5e72003cf978cd3ff13f370008dac14390187cd9757f774448a99
07-06 06:36:59.603 32070-781/? I/PrettyLogger: 
07-06 06:37:00.549 32070-781/? E/PrettyLogger: 
07-06 06:37:00.549 32070-781/? E/PrettyLogger: Thread: Thread-1403
07-06 06:37:00.549 32070-781/? E/PrettyLogger: 
07-06 06:37:00.549 32070-781/? E/PrettyLogger: ByteArrayRequest.<init> (ByteArrayRequest.java:41)
07-06 06:37:00.549 32070-781/? E/PrettyLogger: TLog.e (TLog.java:97)
07-06 06:37:00.549 32070-781/? E/PrettyLogger: 
07-06 06:37:00.549 32070-781/? E/PrettyLogger: Url: https://ap.jawalife.net/jawa/login.do
07-06 06:37:00.549 32070-781/? E/PrettyLogger: 
urlParams:{data
=5FB1A533DC3339C032537048B2EA8EFA389F83233BCA3C85E9FA102436524C584D346ADBB630AE473A3814BB7786E61B1C1F4389A324552DE5EB6B34B450E71EFC2E4AA10900542981BC6170C8B55CAD05
61258B516B868047681580B922E0BE06D06A12C924535670FB794851115A104CB86A09D39A0D15BE649B60A63B54D74B6880CD618EEAC20A1BCAF490DCF6F477E19282E840E76417C3893D9985E7A00C
```

Bottom navigation bar: TODO, Problems, Terminal, Logcat (selected), Profiler, Database Inspector, Event Log, Layout Inspector.

# Live Log Analysis: App Startup/Login

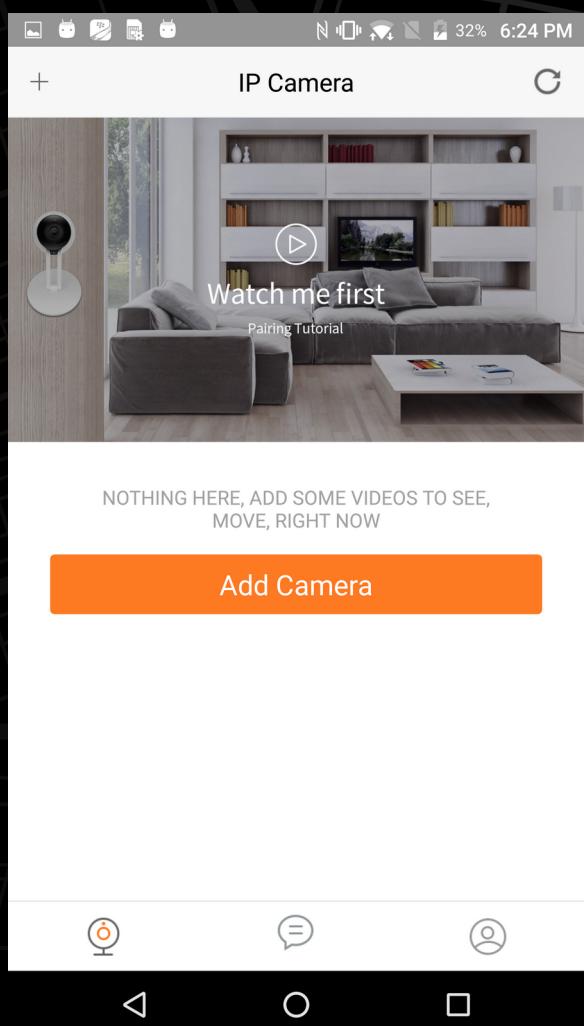
On startup, we find log entries that could be useful:

- Distinctive characters: `biz` 加密前
- URL: <https://ap.jawalife.net/jawa/login.do>
- UrlParams with a lot of data
- And a lot more log messages off screen

Let's see what else turns up

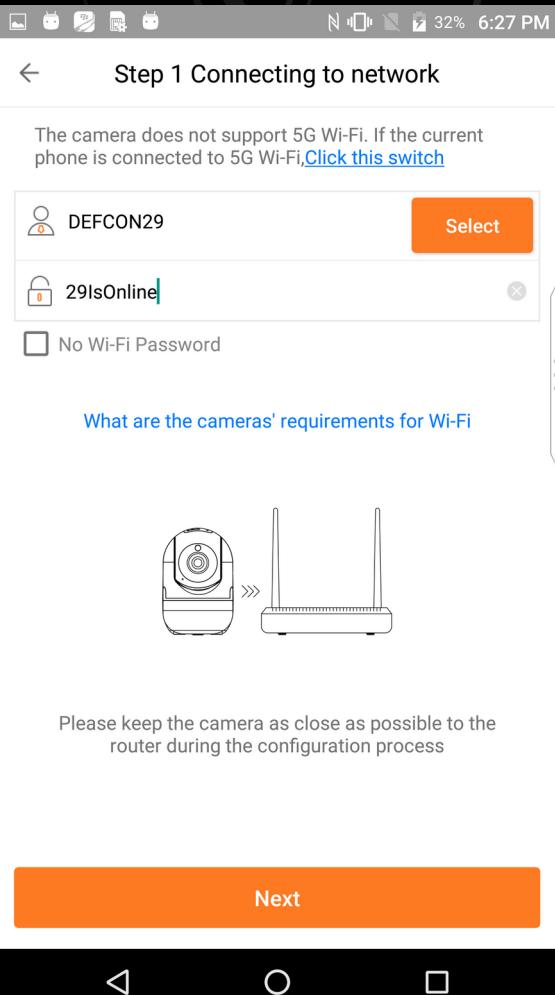


# Logs on Camera Pairing



# Live Log Analysis

## Let's see what we see!



# Live Log Analysis

## Let's see what we see!

# Live Log Analysis: Pairing

A few areas standout

Logcat

Blackberry STV100-1 Android 6.0 ▾ No debuggable processes ▾ Verbose ▾ OutAudio|PRETTYLOGGER[java]Decode ▾ Regex Show only selected application

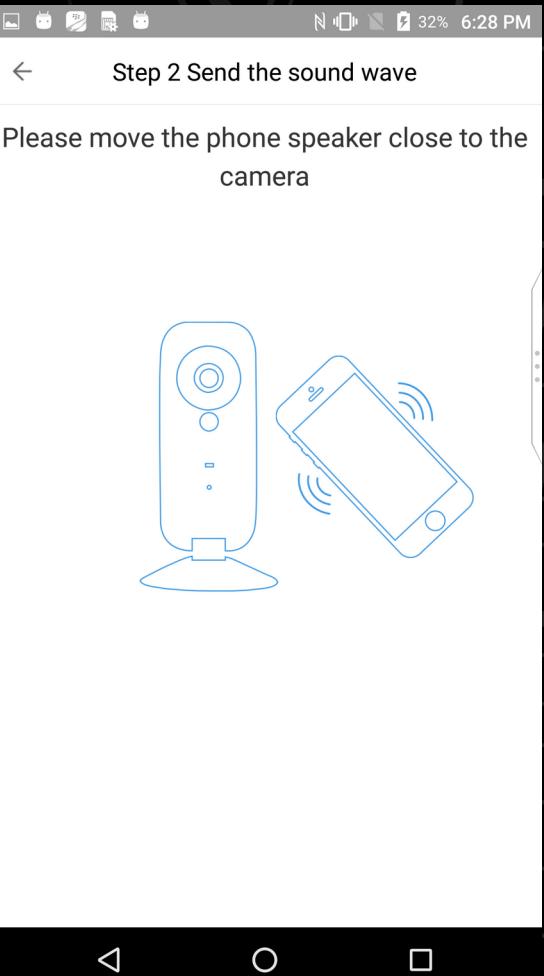
logcat

```
07-06 07:18:40.627 4196-4245/? I/ActivityManager: Displayed com.java.cam/com.ithink.activity.camera.TimeZoneActivity: +259ms
07-06 07:18:47.463 4196-4207/? I/ActivityManager: START u0 {cmp=com.java.cam/com.ithink.activity.camera.BindDeviceNewActivity (has extras)} from uid 10124 on display 0
07-06 07:18:47.821 3461-5324/? I/ExtendedNuUtils: printFileName fd(52) -> /data/app/com.java.cam-1/base.apk
07-06 07:18:47.864 32070-32070/? I/PrettyLogger: Thread: main
07-06 07:18:47.871 32070-32070/? I/PrettyLogger: BindDeviceNewActivity.getUserinfo (BindDeviceNewActivity.java:1205)
07-06 07:18:47.871 32070-32070/? I/PrettyLogger: TLog.i (TLog.java:63)
07-06 07:18:47.871 32070-32070/? I/PrettyLogger: 声波信息->DEFCON29@29isOnline@87g811
07-06 07:18:47.871 32070-32070/? I/PrettyLogger:
07-06 07:18:47.885 32070-32070/? W/System.err: Caused by: java.lang.ClassNotFoundException: Didn't find class "android.view.MiuWindowManager$LayoutParams" on path: DexPathList[[zip file "/data/app/com.java.cam-1/base.apk"],nativeLibraryDirectories=[/data/app/com.java.cam-1/lib/arm64, /data/app/com.java.cam-1/base.apk!/lib/arm64-v8a, /vendor/lib64, /system/lib64]]
07-06 07:18:47.975 4196-4245/? I/ActivityManager: Displayed com.java.cam/com.ithink.activity.camera.BindDeviceNewActivity: +451ms
07-06 07:18:50.889 32070-3324/? I/PrettyLogger: Thread: Thread-1425
07-06 07:18:50.889 32070-3324/? I/PrettyLogger:
07-06 07:18:50.889 32070-3324/? I/PrettyLogger: HttpRequestHelper.getBiz (HttpRequestHelper.java:250)
07-06 07:18:50.898 32070-3324/? I/PrettyLogger: TLog.i (TLog.java:63)
07-06 07:18:50.898 32070-3324/? I/PrettyLogger:
07-06 07:18:50.898 32070-3324/? I/PrettyLogger: biz加密前=name:Java&uid:yeahthisworked@mailinator.com
.comZone=MST&token=qwyBVrAaztVC9LSJBm8kpMVRnkxa0TrYzfwx3&code=87g81&mac=r0t5y9bm3q0qywstya&mac:r0t5y9bm3q0qywstya&exp:010001&mod:00e13bc7761d51da4515ec3994ff4488b7b6b2a5df868fa4143a6edac1dd820236f54b76eaf83ab13fc2692958bd4acf7e629ec984252b9089d70e94c809555af19dc5697dcf253eb1fb352e5cca16c10ab34cf5fe0999fbfe2540168a337c0b805d5e72003cf970cd3ff13f570008dac14390187cd9757f774448a99
07-06 07:18:50.898 32070-3324/? I/PrettyLogger:
```

Event Log Layout Inspector

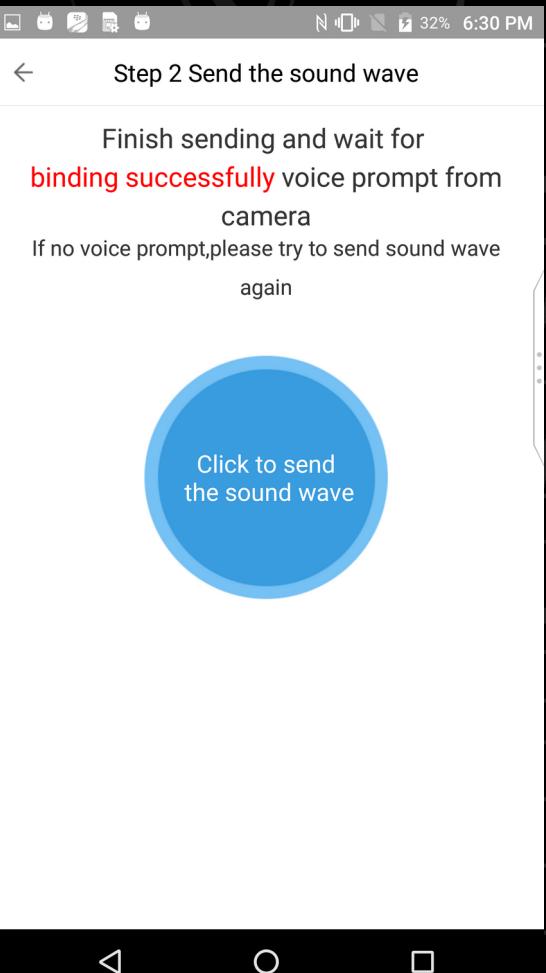
# Live Log Analysis

Let's see what we see!



# Live Log Analysis

Let's see what we see!



# Live Log Analysis: Pairing - Sound Waves

Doesn't look like much? Think again!

```
Logcat
Blackberry STV100-1 ▾ No debuggable proces: ▾ Ver... ▾ OutAudio|PRETTYL × ✓ Regex Show only selected app ▾

logcat
07-06 09:07:05.087 32070-4803/? E/OutAudio: myList size==130095
07-06 09:07:05.134 32070-4803/? E/OutAudio: myList size==58560
07-06 09:07:05.178 32070-4803/? E/OutAudio: myList size==58560
07-06 09:07:05.978 32070-4804/? E/PrettyLogger:
07-06 09:07:05.978 32070-4804/? E/PrettyLogger: || Thread: Thread-1433
07-06 09:07:05.978 32070-4804/? E/PrettyLogger:
07-06 09:07:05.978 32070-4804/? E/PrettyLogger: || ByteArrayRequest.<init> (ByteArrayRequest.java:41)
07-06 09:07:05.978 32070-4804/? E/PrettyLogger: || TLog.e (TLog.java:97)
07-06 09:07:05.978 32070-4804/? E/PrettyLogger:
07-06 09:07:05.978 32070-4804/? E/PrettyLogger: || Url: https://ap.jawalife.net/jawa/checkBindVoice.do
07-06 09:07:05.978 32070-4804/? E/PrettyLogger:
urlParams:{data
=FC5E9020F76F8B8B205298934CCBB57FAF4335275C555745E0AFBCBEF3CD787776F332B5A1D0AD02758ABB941F03E7CD36E7A20C0898EF
30176DE7A10E44D0F897573FEB280547249CF477386163218E0CA49F7FFD2A2128ADE000017B74C6DA622629A3F5D02DC835D3B1AC10B4D
```

Notable String: myList size

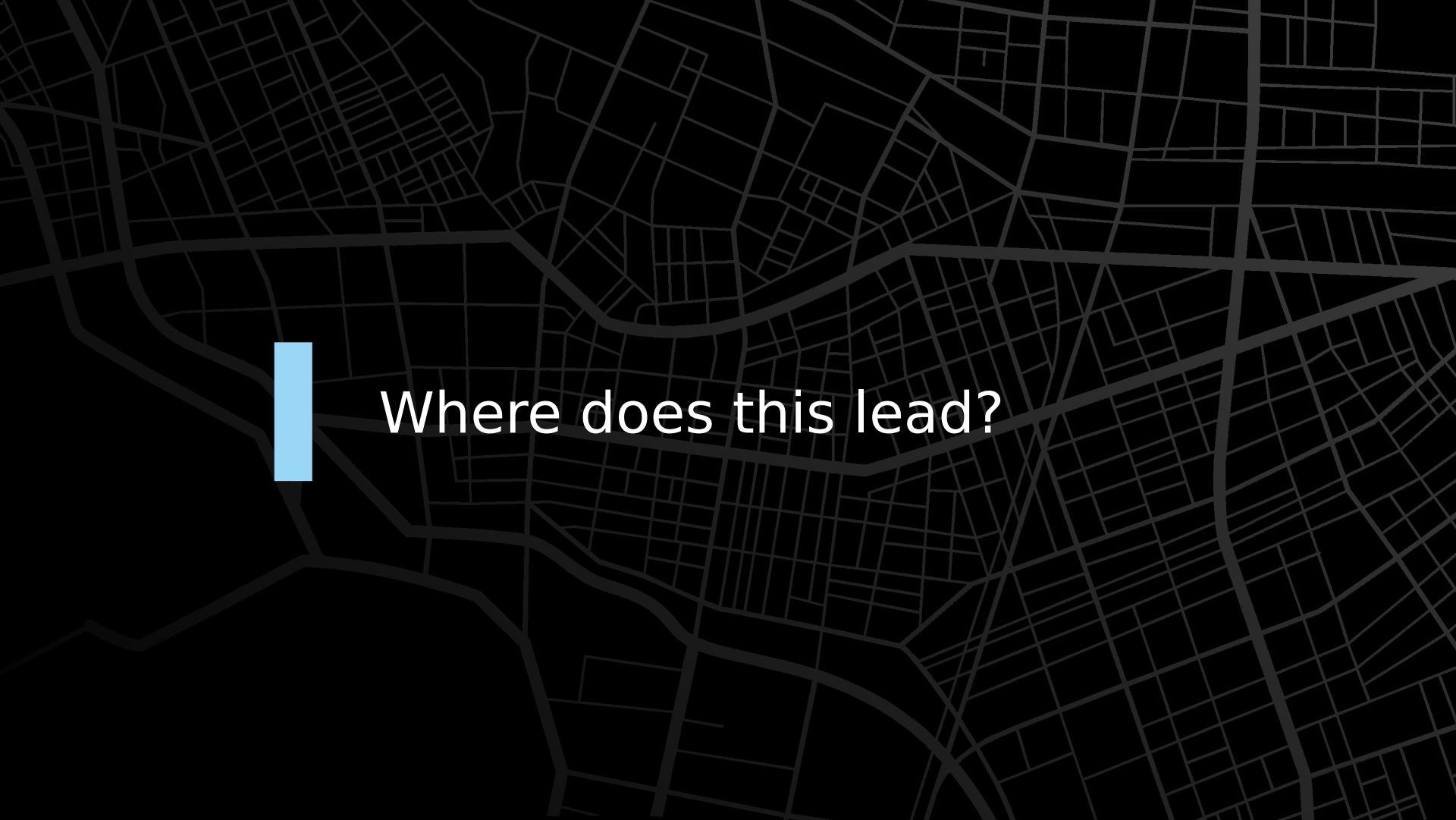
A couple of other items,  
probably more related to  
phoning home:

- ByteArrayRequest
- New URL endpoint  
'checkBindVoice.do'

# Live Log Analysis: Pairing

During the pairing process we discovered

- More distinctive characters, followed by the SSID, Password and... something else?:
  - 声波信息—>DEFCON29-29IsOnline-87gz811
- A class to investigate:
  - com.ithink.activity.camera.BindDeviceNewActivity
- An **HttpRequestHelper** used to phone-home with details about the pairing session
  - The request includes the email associated with my 'cloud account'



Where does this lead?

# Search for Strings

The screenshot shows a Java decompiler interface with a search results window. The search term is '声波信息'. The results show a method in `BindDeviceNewActivity.java` at line 860:

```
TLog.i("声波信息->" + str);
```

The screenshot shows the same Java decompiler interface with the code block expanded. The code is as follows:

```
private String getUserinfo() {
    String str = this.netName + "" +
        (char) 1 +
        this.netPassWord +
        "" +
        (char) 1 +
        this.smartCode +
        MessageService.MSG_DB_NOTIFY_REACHED;

    TLog.i( str: "声波信息->" + str);
    return str;
}
```

com.ithink.activity.camera.BindDeviceNewActivity

- Searching within the project for 声波信息 uncovered a method
- Aside from the expected (SSID/Password), we see:
  - 0x01 used to delimit fields
  - A smartCode (?)
  - The string "1" appended at the end (`MSG_DB_NOTIFY_REACHED = "1"`)

My guess is this is a decompilation artifact

# Search for Strings: What is a SmartCode?

It appears to be randomly-generated for each pairing attempt in `getBundleExtras(Bundle bundle)`

```
BindDeviceNewActivity.java × MessageService.java × HttpRequestHelper.java ×
363     /* access modifiers changed from: protected */
364
365     @Override // com.ithink.lib.base.BaseAppCompatActivity
366     public void getBundleExtras(Bundle bundle) {
367         this.uid = BaseApplication.getInstance().getUserId();
368         this.type = bundle.getString( key: "type");
369         this.netName = bundle.getString( key: "name");
370         this.netPassWord = bundle.getString( key: "pass");
371         this.smartCode = getCharAndNum( i: 6);
372         this.provingNumber = getRandom( i: 0, i2: 32767);
373         this.nickName = getString(R.string.camere_default_name);
374         this.timeZoneId = bundle.getString( key: "timeZone");
375         String str = this.timeZoneId;
376         if (str == null || str.equals("")) {
377             this.timeZoneId = TimeZone.getDefault().getID();
378         }
}
```

# Search for Strings: What is a SmartCode?

Yes, but what \_is\_ a smartCode?

- It's a random code, **used by the camera when it phones home to link up to our cloud profile**

- `this.smartCode = getCharAndNumr(6);`
- Definition of `getCharAndNumr`:

```
private String getCharAndNumr(int i) {  
    Random random = new Random();  
    String str = "";  
    for (int i2 = 0; i2 < i; i2++) {  
        String str2 = random.nextInt(bound: 2) % 2 == 0 ? "char" : "num";  
        if ("char".equalsIgnoreCase(str2)) {  
            str = str + ((char) (random.nextInt(bound: 26) + 97));  
        } else if ("num".equalsIgnoreCase(str2)) {  
            str = str + String.valueOf(random.nextInt(bound: 10));  
        }  
    }  
    return str;  
}
```

# Search for Strings: What Else?



Find in Path 2 matches in 1 file

File mask:  ↻ Aa W.\*

Q~ myList size

In Project Module Directory Scope e/riion/tmp/apk/java-decompiled ...

Log.e("OutAudio", "myList size==" + this.myList\_v3s.size()), OutAudio.java 130

String str = myList size==";", OutAudio.java 168

```
OutAudio.java sources/com/ithink/voice
127
128     for (int i2 = 0; i2 < this.myList_v3s.size(); i2++) {
129         sArr[i2] = this.myList_v3s.get(i2).shortValue();
130     }
131     Log.e( tag: "OutAudio", msg: "myList size==" + this.myList_v3s.size());
132     Message obtainMessage = this.handler.obtainMessage();
133     obtainMessage.arg2 = (this.myList_v3s.size() * 1000) / Audi
134     obtainMessage.what = R.id.PLAYVOICE_PROGRESS;
135     this.handler.sendMessage(obtainMessage);
136 }
```

Ctrl+Enter Open in Find Window

BindDeviceNewActivity.java x OutAudio.java x

34 ▲ 44 ▼ 14 ^

```
BindDeviceNewActivity.java
150
151     private void playVoice() {
152         this.trackplayer_v3s = new AudioTrack( streamType: 3, this.voiseStruct.getAud
153         this.trackplayer_v3s.play();
154         int length = (this.ssid.length() + this.pass.length() + 6 + 3) * 4;
155         int[] iArr = new int[length];
156         int voiseStructV3s = this.voiseStruct.getVoiseStructV3s(this.ssid.getBytes()

OutAudio.java
112 ⚡ ⚡
113     public void run() {
114         if (this.ssid == null) {
115             this.trackplayer_v3s = new AudioTrack( streamType: 3, AudioRegCommon.SAMP
116             this.trackplayer_v3s.play();
117             play1Atom(AudioRegCommon.FREQ_BEGIN, AudioRegCommon.SAMPLE_FREQ, AudioR
```

Check another string from log analysis: **myList size**

It looks like the magic happens in these 2 methods:

- **run()**
- **playVoice()**



# Extractive Analysis

# Extractive Analysis

At this point we've learned a lot about the codebase

We can **extract** the code we need to automate our analysis  
and place it into a **clean project**

# Extractive Analysis

A couple of notes on project setup

- You may need to manually create your **jniLibs** folder
  - Inside you'll need to place all the contents of the donor project's lib/jniLibs directory
- The **package/class name has to match** what you pull from the donor project, or it won't load

Once configured, you'll be able to automate a black-box analysis of the compiled library

# A Question Answered

```
System.out.println("Initiated mapNum2Freq");
}

public int getFreqFromNum(int i) {
    Init();
    Integer num = this.mapNum2Freq.get(Integer.valueOf(i));
    if (num != null) {

}

```

NING

otivator\_reverse\_jawa)

example.badmotivator\_

xample.badmotivator\_re

nple.badmotivator\_re

widget)

n.google.android.materi

id.view)

view)

s)

os)

Caller (com.android.inter

rnal.os)

Variables

+ - this = {VCodeTable@32621}

- f mapNum2Freq = {HashMap@32622} size = 16

- {Integer@32642} 0 -> {Integer@32643} 3000
- {Integer@32644} 1 -> {Integer@32645} 3125
- {Integer@32646} 2 -> {Integer@32647} 3250
- {Integer@32648} 3 -> {Integer@32649} 3375
- {Integer@32650} 4 -> {Integer@32651} 3500
- {Integer@32652} 5 -> {Integer@32653} 3625
- {Integer@32654} 6 -> {Integer@32655} 3750
- {Integer@32656} 7 -> {Integer@32657} 3875
- {Integer@32658} 8 -> {Integer@32659} 4000
- {Integer@32660} 9 -> {Integer@32661} 4125
- {Integer@32662} 10 -> {Integer@32663} 4250
- {Integer@32664} 11 -> {Integer@32665} 4375
- {Integer@32666} 12 -> {Integer@32667} 4500
- {Integer@32668} 13 -> {Integer@32669} 4625
- {Integer@32670} 14 -> {Integer@32671} 4750
- {Integer@32672} 15 -> {Integer@32673} 4875

VCodeTable holds the key to digit/frequency mapping

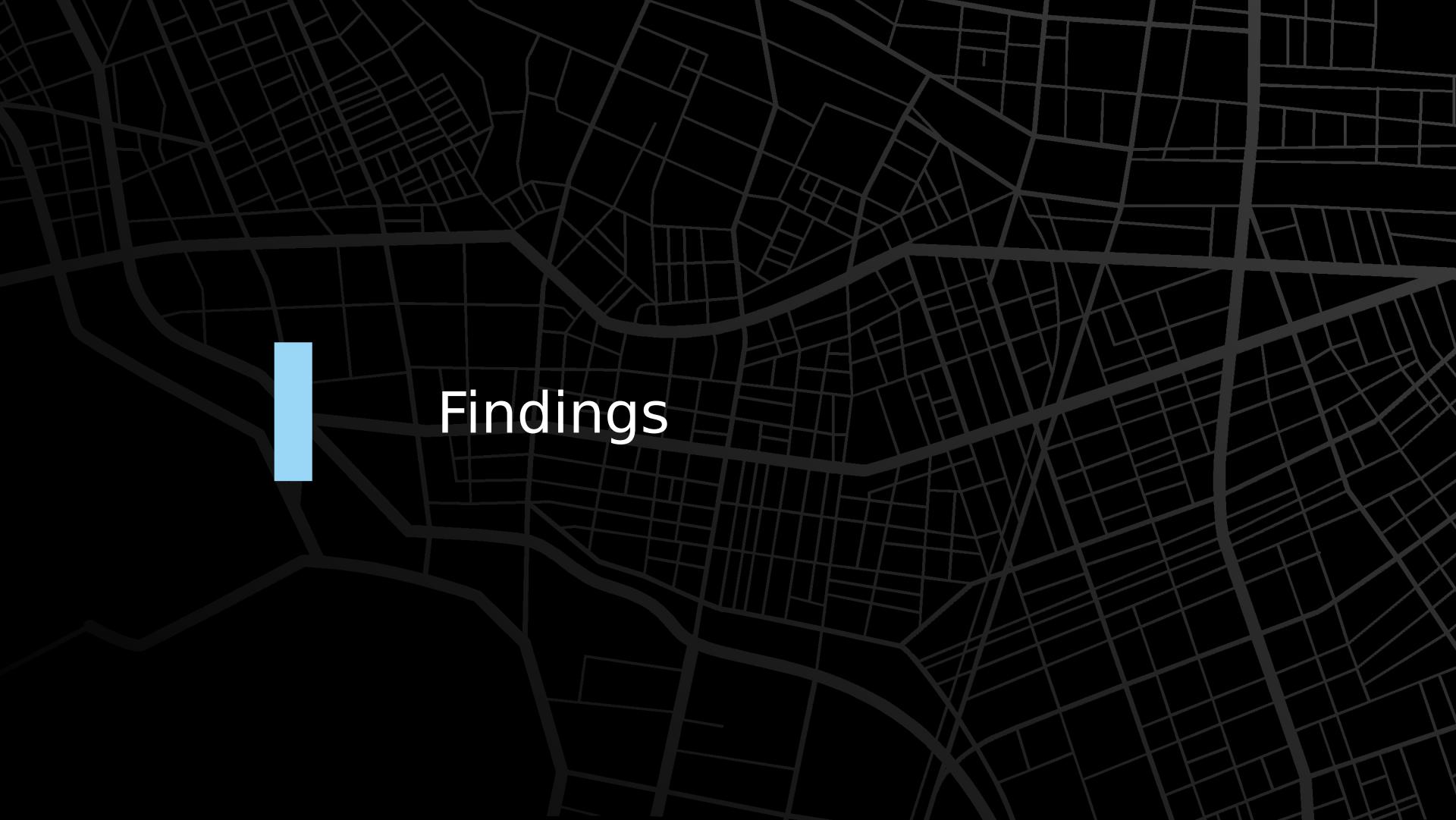
We found this by extracting the class into a test project and executing it.

# A Question Answered

0=3000Hz	8=4000Hz
1=3125Hz	9=4125Hz
2=3250Hz	A=4250Hz
3=3375Hz	B=4375Hz
4=3500Hz	C=4500Hz
5=3625Hz	D=4625Hz
6=3750Hz	E=4750Hz
7=3875Hz	F=4875Hz

`VCodeTable` holds the key to digit/frequency mapping

We found this by extracting the class into a test project and executing it.



# Findings

# Findings

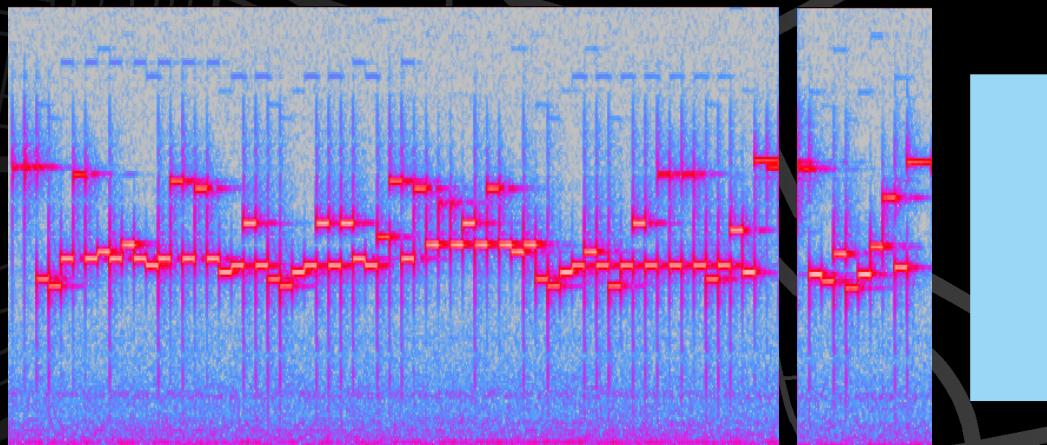
Starting in `com.ithink.voice.OutAudio` and tracing, we find:

1. `AudioTrack` to play the signal
  - Appears to be 3 blocks to a single transmission
2. `strInput == getUserinfo()`, which contains:
  - SSID, Password, Random 'smartcode'
3. Control tones:
  - `AudioRegCommon.FREQ_BEGIN` and `FREQ_END`
4. Space Tone is used to split up identical tones
  - Ensures the receiver can discern between 2 distinct tones and not one 'long' tone
5. `play1Char` which points us to:
  - `VcodeTable` class which maps digits to frequency tones. Looks like Hex encoding!
6. The use of `CRC` values
  - `GetCrcVal(String str)` uses a `shifted CRC32` for error correction
7. An opaque `VoiceStruct` class that loads a library by the same name
  - If we need to drill-down into this library we will require a binary decompiler

# Findings

Using what we know we can:

- Reconstruct Section 1
- Reconstruct Section 2



Section 1

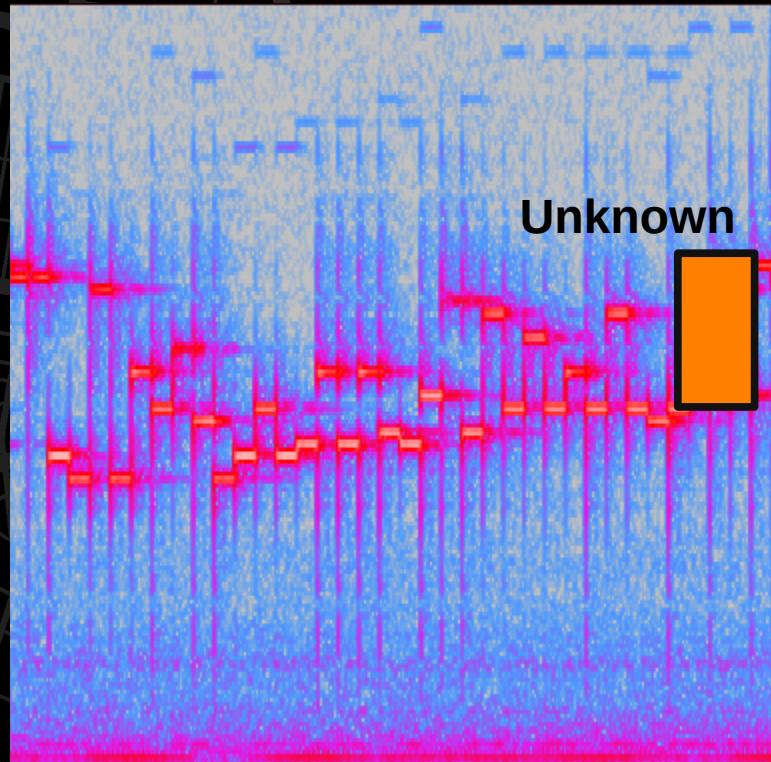
Section 2

# Findings

Section 3 is incomplete

1. We know everything except for one signal in section 3
  - This appears to be an error correction code of some kind(?)

To uncover the remaining secrets of section 3, we need binary analysis





# GHIDRA

Signal blocks 2 and 3 are generated by [VoiseStruct](#). Reversing compiled code requires a binary analysis tool. [Ghidra](#) is the tool I'll be using





A faint, grayscale map of a city's street grid serves as the background for the slide. The grid consists of numerous intersecting streets, creating a pattern of rectangular blocks. Some major roads are highlighted in a darker shade of gray, forming a network that radiates from the center of the city.

# Get Setup with Ghidra



# Get Setup with Ghidra

- Visit <https://github.com/NationalSecurityAgency/ghidra/releases>
  - Pull the **latest version**
- Follow the installation guidance for your platform:
  - <https://github.com/NationalSecurityAgency/ghidra#install>

# Get Setup with Ghidra

- Open a new project
  - File -> New Project -> Fill out wizard
  - Click the 
- Import the library:
  - File -> Import File
  - jawa-decompiled/resources/lib/x86\_64/libvoisestruct.so
  - Import with default options
  - Select Yes to analyze the file



RE With Ghidra

# RE With Ghidra

Identify which Library functions are used

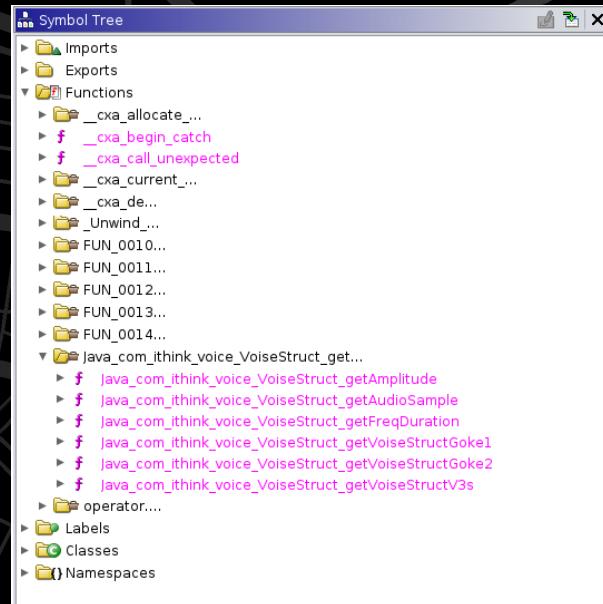
- From within Android Studio we find these functions are used

```
int voiceStructV3s = this.voiceStruct.getVoiceStructV3s  
int voiceStructGoke1 = this.voiceStruct.getVoiceStructGoke1  
int voiceStructGoke2 = this.voiceStruct.getVoiceStructGoke2
```

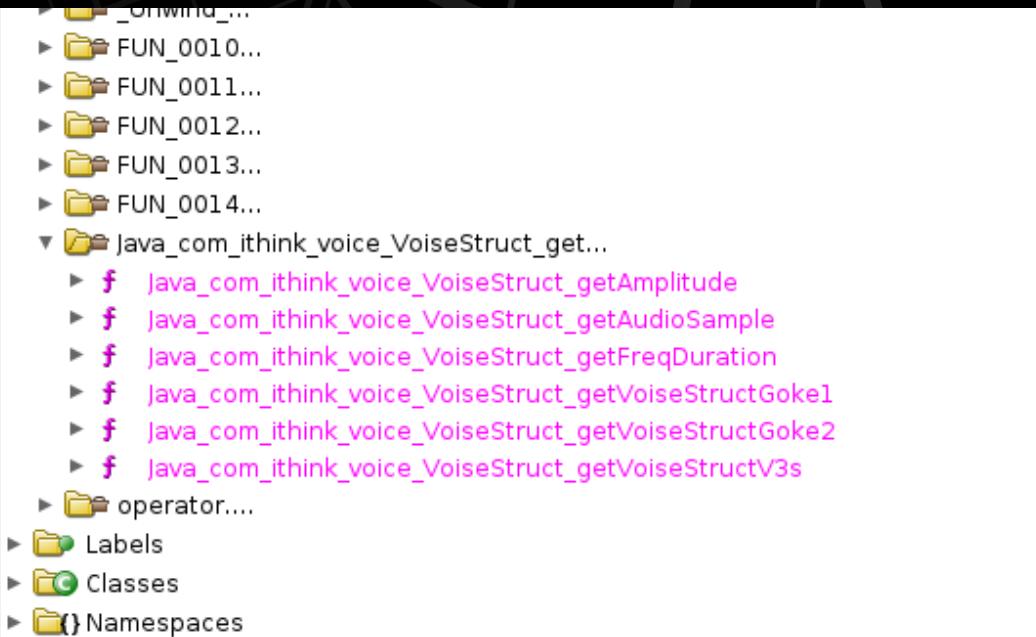
# RE With Ghidra

## Investigate the Library

- Examine the **Symbol Tree** to find the **JNI** functions
  - Java\_com\_ithink\_voice\_VoiseStruct\_getAmplitude
  - Java\_com\_ithink\_voice\_VoiseStruct\_getAudioSample
  - Java\_com\_ithink\_voice\_VoiseStruct\_getFreqDuration
  - Java\_com\_ithink\_voice\_VoiseStruct\_getVoiseStructGoke1
  - Java\_com\_ithink\_voice\_VoiseStruct\_getVoiseStructGoke2
  - Java\_com\_ithink\_voice\_VoiseStruct\_getVoiseStructV3s



# RE With Ghidra



The screenshot shows the Ghidra interface with a file tree on the left. The tree includes:

- ▶ \_Onwind\_...
- ▶ FUN\_0010...
- ▶ FUN\_0011...
- ▶ FUN\_0012...
- ▶ FUN\_0013...
- ▶ FUN\_0014...
- ▶ Java\_com\_ithink\_voice\_VoiseStruct\_get...
  - ▶ f Java\_com\_ithink\_voice\_VoiseStruct\_getAmplitude
  - ▶ f Java\_com\_ithink\_voice\_VoiseStruct\_getAudioSample
  - ▶ f Java\_com\_ithink\_voice\_VoiseStruct\_getFreqDuration
  - ▶ f Java\_com\_ithink\_voice\_VoiseStruct\_getVoiseStructGoke1
  - ▶ f Java\_com\_ithink\_voice\_VoiseStruct\_getVoiseStructGoke2
  - ▶ f Java\_com\_ithink\_voice\_VoiseStruct\_getVoiseStructV3s
- ▶ operator....
- ▶ Labels
- ▶ Classes
- ▶ Namespaces

A Closer Look at what we see in Ghidra

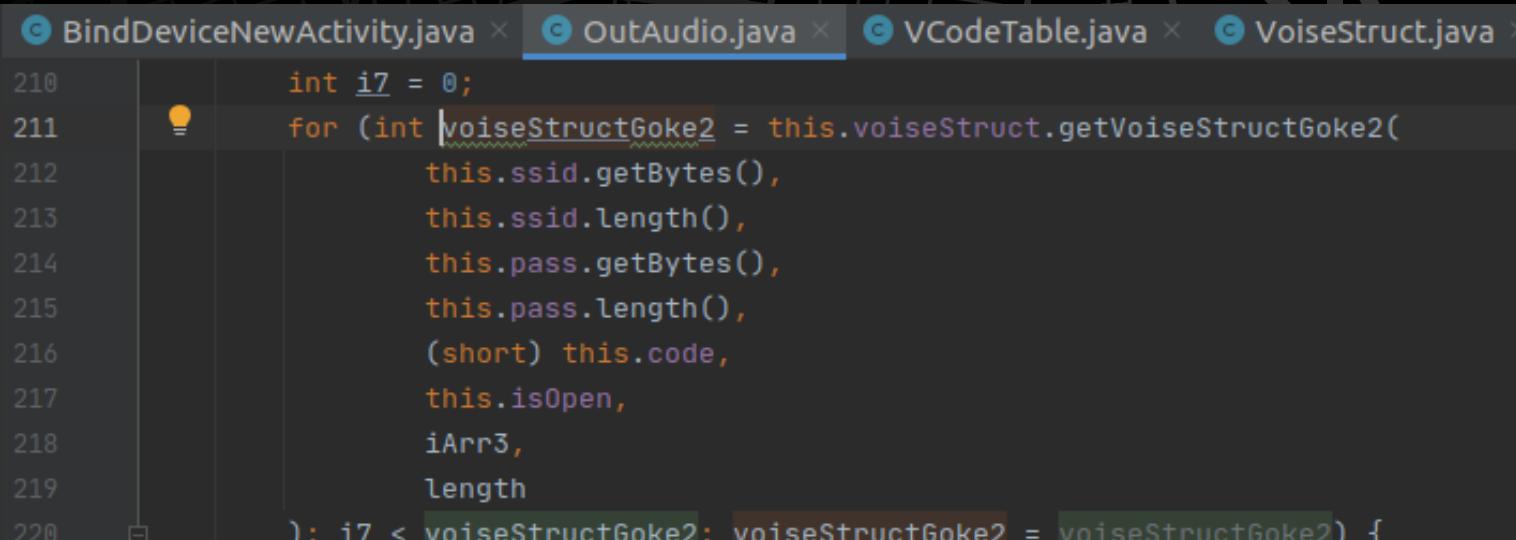


Pick a Function and Dig In

# Ghidra: Analyze a Function

Key function: `getVoiseStructGoke2`

- Generates Signal sections 2 and 3
- Here's what we see in Android Studio:
  - 8 arguments passed to `getVoiseStructGoke2`

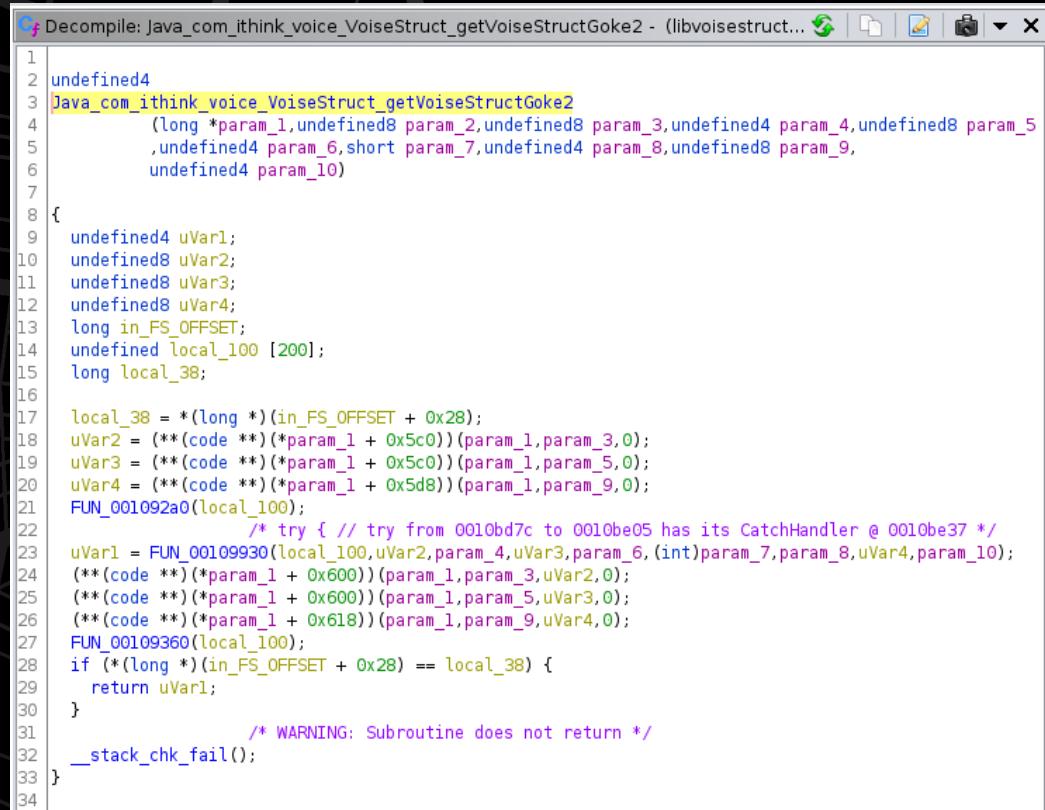


```
BindDeviceNewActivity.java × OutAudio.java × VCodeTable.java × VoiseStruct.java ×
210             int i7 = 0;
211             for (int voiseStructGoke2 = this.voiseStruct.getVoiseStructGoke2(
212                     this.ssid.getBytes(),
213                     this.ssid.length(),
214                     this.pass.getBytes(),
215                     this.pass.length(),
216                     (short) this.code,
217                     this.isOpen,
218                     iArr3,
219                     length
220             ); i7 < voiseStructGoke2; voiseStructGoke2 = voiseStructGoke2) {
```

# Ghidra: Analyze a Function

## Ghidra: getVoiseStructGoke2

- Appears to be a parameter count mis-match. This is related to JNI
  - 8 params passed from Java
  - 10 params in Ghidra



The screenshot shows the Ghidra decompiler window with the following Java code:

```
C:\Decompile\java_com_ithink_voice_VoiseStruct_getVoiseStructGoke2 - (libvoisestruct...)
```

```
1 undefined4
2
3 Java_com_ithink_voice_VoiseStruct_getVoiseStructGoke2
4     (long *param_1,undefined8 param_2,undefined8 param_3,undefined4 param_4,undefined8 param_5
5      ,undefined4 param_6,short param_7,undefined4 param_8,undefined8 param_9,
6      undefined4 param_10)
7
8 {
9     undefined4 uVar1;
10    undefined8 uVar2;
11    undefined8 uVar3;
12    undefined8 uVar4;
13    long in_FS_OFFSET;
14    undefined local_100 [200];
15    long local_38;
16
17    local_38 = *(long *) (in_FS_OFFSET + 0x28);
18    uVar2 = (**(code **)(*param_1 + 0x5c0))(param_1,param_3,0);
19    uVar3 = (**(code **)(*param_1 + 0x5c0))(param_1,param_5,0);
20    uVar4 = (**(code **)(*param_1 + 0x5d8))(param_1,param_9,0);
21    FUN_001092a0(local_100);
22
23    /* try { // try from 0010bd7c to 0010be05 has its CatchHandler @ 0010be37 */
24    uVar1 = FUN_00109930(local_100,uVar2,param_4,uVar3,param_6,(int)param_7,param_8,uVar4,param_10);
25    (**(code **)(*param_1 + 0x600))(param_1,param_3,uVar2,0);
26    (**(code **)(*param_1 + 0x600))(param_1,param_5,uVar3,0);
27    (**(code **)(*param_1 + 0x618))(param_1,param_9,uVar4,0);
28    FUN_00109360(local_100);
29    if (*(long *) (in_FS_OFFSET + 0x28) == local_38) {
30        return uVar1;
31    }
32    /* WARNING: Subroutine does not return */
33    __stack_chk_fail();
34}
```



# A Note on JNI

# Digging Deeper: A Word on JNI

The argument-count did not line up between Android and the JNI library call

This is due to **JNI Calling Conventions**

- There are **2 additional arguments** as part of the specification:
  - JNIEnv pointer
  - jobject
- These 2 params are **front-loaded** to the function signature

# Digging Deeper: A Word on JNI

Raw decompiled view

```
undefined4
Java_com_ithink_voice_VoiceStruct_getVoiceStructGoke2
    (long *param_1,undefined8 param_2,undefined8 param_3,undefined4 param_4,undefined8 param_5
 ,undefined4 param_6,short param_7,undefined4 param_8,undefined8 param_9,
 undefined4 param_10)
```

Refactored in Ghidra

```
undefined4
Java_com_ithink voice VoiceStruct getVoiceStructGoke2
    [REDACTED] undefined8 ssidB,undefined4 ssidBLen,undefined8 passB,
    undefined4 passBLen,short smartCode,undefined4 isNetworkOpen,undefined8 returnArr,
    undefined4 returnArrLen)
```

\* I couldn't rename param\_2 to jobject as it's not actually used in the library call



Continue the Analysis

# Ghidra: Analyze a Function

CodeBrowser: Java\_com\_ithink\_voice\_VoiceStruct\_getVoiceStructGoKe2 - (libvoisestruct.so)

File Edit Analysis Graph Navigation Search Select Tools Window Help

I D U L F R B V C f G S P

Programs X Listing: Java\_com\_ithink\_voice\_VoiceStruct\_getVoiceStructGoKe2 - (libvoisestruct.so) X

Programs X Symb... X

Imports Exports Functions Labels Classes Namespaces

Filter: D... X

Console - Scripting

0010bcf0 java\_com\_ithink\_v... PUSH RBP

```
1 undefined4
2 Java_com_ithink_voice_VoiceStruct_getVoiceStructGoKe2
3     (long *param_1,undefined8 param_2,undefined8 param_3,undefined4 param_4,undefined8 param_5
4     ,undefined4 param_6,short param_7,undefined4 param_8,undefined8 param_9,
5     undefined4 param_10)
6
7 {
8     undefined4 uVar1;
9     undefined8 uVar2;
10    undefined8 uVar3;
11    undefined8 uVar4;
12    long in_FS_OFFSET;
13    undefined local_100 [200];
14    long local_38;
15
16    local_38 = *(long *)(in_FS_OFFSET + 0x28);
17    uVar2 = (**(code **)(param_1 + 0x5c0))(param_1,param_3,0);
18    uVar3 = (**(code **)(param_1 + 0x5c0))(param_1,param_5,0);
19    uVar4 = (**(code **)(param_1 + 0x5d8))(param_1,param_9,0);
20    FUN_001092a0(local_100);
21        /* try { // try from 0010bd7c to 0010be05 has its CatchHandler @ 0010be37 */
22    uVar1 = FUN_00109930(local_100,uVar2,param_4,uVar3,param_6,(int)param_7,param_8,uVar4,param_10);
23    (**(code **)(param_1 + 0x6000))(param_1,param_3,uVar2,0);
24    (**(code **)(param_1 + 0x6000))(param_1,param_5,uVar3,0);
25    (**(code **)(param_1 + 0x618))(param_1,param_9,uVar4,0);
26    FUN_00109360(local_100);
27    if (*(long *)(in_FS_OFFSET + 0x28) == local_38) {
28        return uVar1;
29    }
30    /* WARNING: Subroutine does not return */
31    _stack_chk_fail();
32}
33
34}
```

Function Decompiler View

# Ghidra: Analyze a Function

CodeBrowser: java-voisestruc-raw-undeciphered:/libvoisestruct.so

gation Search Select Tools Window Help

I D U L F R V B

C Decompile: Java\_com\_ithink\_voice\_VoiceStruct\_getVoiceStructGoke2 - (libvoisestruct.so)

```
1 undefined4
2 Java_com_ithink_voice_VoiceStruct_getVoiceStructGoke2
3     (long *param_1,undefined8 param_2,undefined8 param_3,undefined4 param_4,undefined8 param_5
4     ,undefined4 param_6,short param_7,undefined4 param_8,undefined8 param_9,
5     undefined4 param_10)
6
7 {
8     undefined4 uVar1;
9     undefined8 uVar2;
10    undefined8 uVar3;
11    undefined8 uVar4;
12    long in_FS_OFFSET;
13    undefined Local_100 [200];
14    long local_38;
15
16
17    local_38 = *(long *)(in_FS_OFFSET + 0x28);
18    uVar2 = (**(code **)(param_1 + 0x5c0))(param_1,param_3,0);
19    uVar3 = (**(code **)(param_1 + 0x5c0))(param_1,param_5,0);
20    uVar4 = (**(code **)(param_1 + 0x5d8))(param_1,param_9,0);
21    FUN_001092a0(local_100);
22    /* try { // try from 0010bd7c to 0010be05 has its CatchHandler @ 0010be37 */
23    uVar1 = FUN_00109930(local_100,uVar2,param_4,uVar3,param_6,(int)param_7,param_8,uVar4,param_10);
24    (**(code **)(param_1 + 0x600))(param_1,param_3,uVar2,0);
25    (**(code **)(param_1 + 0x600))(param_1,param_5,uVar3,0);
26    (**(code **)(param_1 + 0x618))(param_1,param_9,uVar4,0);
27    FUN_00109360(local_100);
28    if ((*((long *)in_FS_OFFSET + 0x28) == local_38) {
29        return uVar1;
30    }
31    /* WARNING: Subroutine does not return */
32    _stack_chk_fail();
33}
34}
```

Continued analysis of  
getVoiceStructGoke2:

- **FUN\_00109930** leverages our inputs
- First, let's cleanup this view...

# Ghidra: Analyze a Function

```
1 undefined4
2 Java_com_ithink_voice_VoiceStruct_getVoiceStructGoke2
3     (long *jni,undefined8 param_2,undefined8 ssidB,undefined4 ssidBLen,undefined8 passB,
4      undefined4 passBLen,short smartCode,undefined4 isNetworkOpen,undefined8 returnArr,
5      undefined4 returnArrLen)
6
7 {
8     undefined4 uVar1;
9     undefined8 ssidBLocal;
10    undefined8 passBLocal;
11    undefined8 returnArrLocal;
12    long in_FS_OFFSET;
13    undefined local_100 [200];
14    long local_38;
15
16    local_38 = *(long *) (in_FS_OFFSET + 0x28);
17    ssidBLocal = (**(code **)(*jni + 0x5c0))(jni,ssidB,0);
18    passBLocal = (**(code **)(*jni + 0x5c0))(jni,passB,0);
19    returnArrLocal = (**(code **)(*jni + 0x5d8))(jni,returnArr,0);
20    InitReturnValues(local_100);
21    /* try { // try from 0010bd7c to 0010be05 has its CatchHandler @ 0010be37 */
22    uVar1 = Section2and3Generator
23        (local_100,ssidBLocal,ssidBLen,passBLocal,passBLen,(int)smartCode,isNetworkOpen,
24         returnArrLocal,returnArrLen);
25    (**(code **)(*jni + 0x600))(jni,ssidB,ssidBLocal,0);
26    (**(code **)(*jni + 0x600))(jni,passB,passBLocal,0);
27    (**(code **)(*jni + 0x618))(jni,returnArr,returnArrLocal,0);
28    ReturnVoid(local_100);
29    if (* (long *) (in_FS_OFFSET + 0x28) == local_38) {
30        return uVar1;
31    }
32    /* WARNING: Subroutine does not return */
33    __stack_chk_fail();
34}
35
36
```

Cleaned-up view of  
getVoiceStructGoke2:

- FUN\_00109930 renamed to `Section2and3Generator`
- Time to dig-deeper
- We want to identify the final code that is missing from our waveform

# Ghidra: Analyze a Function

## Raw view of Section2and3Generator

- Kind of terse
- We need more descriptive identifiers

```
C:\Decompile: FUN_00109930 - (libvoisestruct.so)
1
2 int FUN_00109930(undefined8 param_1,undefined8 param_2,undefined4 param_3,undefined *param_4,
3                     int param_5,undefined4 param_6,int param_7,long param_8)
4
5 {
6     undefined uVar1;
7     undefined2 uVar2;
8     int iVar3;
9     int iVar4;
10    size_t _n;
11    undefined *_src;
12    long in_FS_OFFSET;
13    undefined local_438;
14    undefined local_437;
15    undefined local_436;
16    undefined local_435;
17    undefined local_434;
18    undefined local_433 [507];
19    undefined local_238;
20    undefined local_237;
21    undefined local_236;
22    undefined local_235;
23    long local_38;
24
25    local_38 = *(long *)(in_FS_OFFSET + 0x28);
26    if (param_7 == 1) {
27        param_5 = param_7;
28    }
29    _src = &DAT_00142940;
30    if (param_7 != 1) {
31        _src = param_4;
32    }
33    memset(&local_238,0,0x200);
34    local_238 = 0x12;
35    local_435 = (undefined)param_6;
36    local_434 = (undefined)((uint)param_6 >> 8);
37    local_237 = local_435;
38    local_236 = local_434;
39    local_235 = FUN_0010a740(&local_238,3);
40    iVar3 = FUN_00109590(param_1,2,&local_238,4,param_8);
41    _n = SEXT48(param_5);
42    memset(&local_438,0,0x200);
43    local_438 = 2;
44    uVar2 = FUN_0010a740(param_2,param_3);
45    local_437 = (undefined)uVar2;
46    local_436 = (undefined)((ushort)uVar2 >> 8);
47    memcpy(local_433,_src,_n);
48    uVar1 = FUN_0010a740(&local_438,_n + 5 & 0xffffffff);
49    local_433[_n] = uVar1;
50    iVar4 = FUN_00109590(param_1,2,&local_438,param_5 + 6,param_8 + (long)iVar3 * 4);
51    if (* (long *) (in_FS_OFFSET + 0x28) == local_38) {
52        return iVar4 + iVar3;
53    }
54    /* WARNING: Subroutine does not return */
55    _stack_chk_fail();
56 }
```

# Ghidra: Analyze a Function

## Section2and3Generator deciphered

- Start with what you know
- Once you find a good starting point, the rest of symbol names fall into place
- There can be value even if you don't get the names 100% correct
- Critical section highlighted, which is what I need to reproduce the signal
- To accurately reproduce the signal, I need to take the CRC value of the SSID and shift it

```
C:\Decompile: Section2and3Generator - (libvoiesstruct.so)

1 int Section2and3Generator
2         (undefined8 returnArr,undefined8 ssidB,undefined4 ssidBLen,undefined *passB,
3          int passBLen,undefined4 smartCode,int isOpenNet,long returnArrLen)
4
5 {
6     undefined iSection3StartValCrcd;
7     undefined2 ssidCrc;
8     int tonesRetVal;
9     int iVar1;
10    size_t _n;
11    undefined *passBLocal;
12    long in_FS_OFFSET;
13    undefined section3StartVal;
14    undefined iSsidCrc;
15    undefined iSsidCrcMangled;
16    undefined smartCodeLocal;
17    undefined smartCodeLocalShifted;
18    undefined iPassBLocal [507];
19    undefined section2StartVal;
20    undefined iSmartCodeLocal;
21    undefined iSmartCodeLocalShifted;
22    undefined iSection2StartValCrcd;
23    long local_38;
24
25    local_38 = *(long *)(in_FS_OFFSET + 0x28);
26    if (isOpenNet == 1) {
27        passBLen = isOpenNet;
28    }
29    passBLocal = &DAT_00142940;
30    if (isOpenNet != 1) {
31        passBLocal = passB;
32    }
33    memset(&section2StartVal,0,0x200);
34    section2StartVal = 0x12;
35    smartCodeLocal = (undefined)smartCode;
36    smartCodeLocalShifted = (undefined)((uint)smartCode >> 8);
37    iSmartCodeLocal = smartCodeLocal;
38    iSmartCodeLocalShifted = smartCodeLocalShifted;
39    iSection2StartValCrcd = CRC(&section2StartVal,_n);
40    tonesRetVal = GenerateTones(returnArr,2,&section2StartVal,4,returnArrLen);
41    __n = SEXT48(passBLen);
42    memset(&section3StartVal,0,0x200);
43    section3StartVal = 2;
44    ssidCrc = CRC(ssidB,ssidBLen);
45    iSSIDCrc = (undefined)ssidCrc;
46
47    iSSIDCrcMangled = (undefined)((ushort)ssidCrc >> 8);
48    memcpy(iPassBLocal,passBLocal,__n);
49    iSection3StartValCrcd = CRC(&section3StartVal,__n + 5 & 0xffffffff);
50    iPassBLocal[__n] = iSection3StartValCrcd;
51    iVar1 = GenerateTones(returnArr,2,&section3StartVal,passBLen + 6,
52                           returnArrLen + (long)tonesRetVal * 4);
53    if (*((long *)in_FS_OFFSET + 0x28) == local_38) {
54        return iVar1 + tonesRetVal;
55    }
56    /* WARNING: Subroutine does not return */
57    _stack_chk_fail();
58 }
```

# Ghidra: Analyze a Function

Zero-in on the section: **Needs another bitshift!**

```
ssidCrcMangled = (undefined)((ushort)ssidCrc >> 8);
```

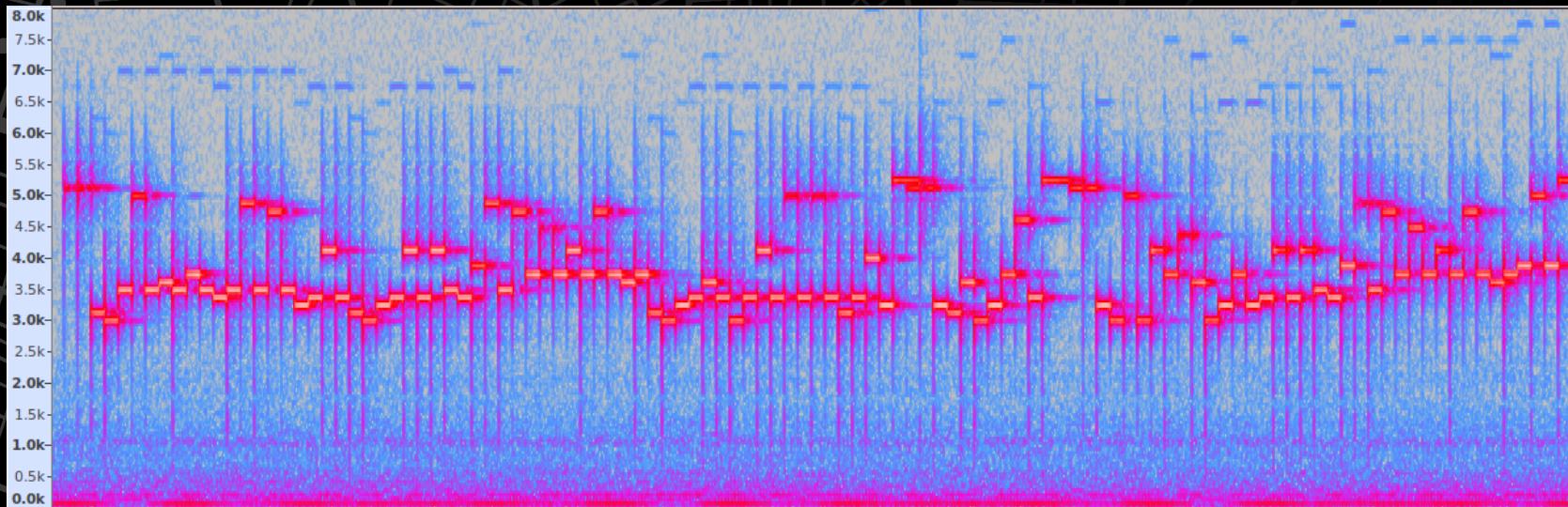
```
44     section3StartVal = 2;
45     ssidCrc = CRC(ssidB,ssidBLen);
46     iSSIDCrc = (undefined)ssidCrc;
47     iSSIDCrcMangled = (undefined)((ushort)ssidCrc >> 8);
48     memcpy(iPassBLocal,passBLocal,__n);
49     iSection3StartValCrcd = CRC(&section3StartVal,__n + 5 & 0xffffffff);
50     iPassBLocal[__n] = iSection3StartValCrcd;
51     iVar1 = GenerateTones(returnArr,2,&section3StartVal.passBLen + 6,
```

04

# Hack the Signal

Can We Recreate It?

Let's look at what we know

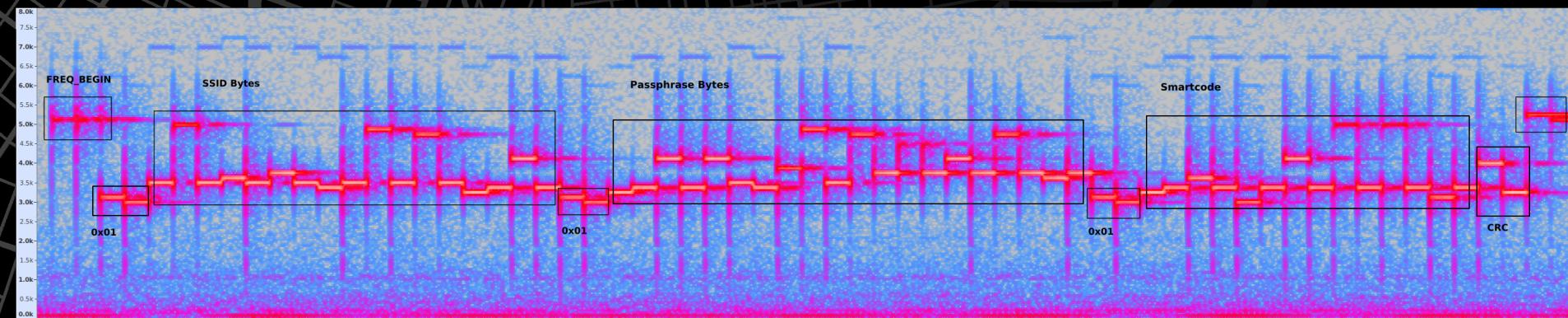


# The Waveform: What we know

1. The waveform is comprised of **3 sections** of hexified data
2. Each section is prefixed and suffixed by **control codes** and **section identifiers**
3. When 2 sequential tones are identical, a **space tone** is used between them
4. The duration of each tone is around **50 / 60 milliseconds**
5. We know the **structure** of each waveform section

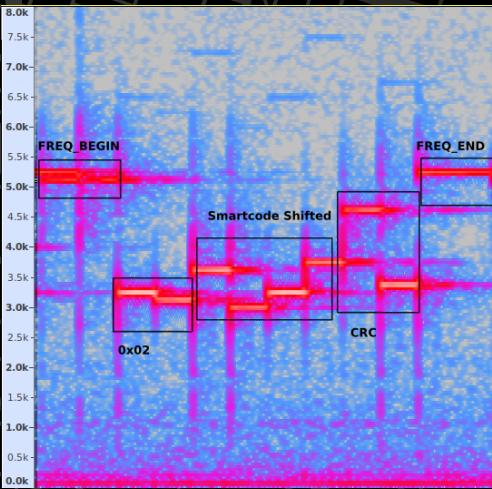
# I Section 1 Signal Structure

1. FREQ\_BEGIN
2. FREQ\_BEGIN
3. 0x01
4. SSID\_BYTES
5. 0x01
6. PASSPHRASE\_BYTES
7. 0x01
8. SMARTCODE\_DIGITS
9. CRC( $0x01 + \text{SSID\_BYTES} + 0x01 + \text{PASSPHRASE\_BYTES} + 0x01 + \text{SMARTCODE\_DIGITS} + "1"$ )
10. FREQ\_END
11. FREQ\_END



# Section 2

# Signal Structure

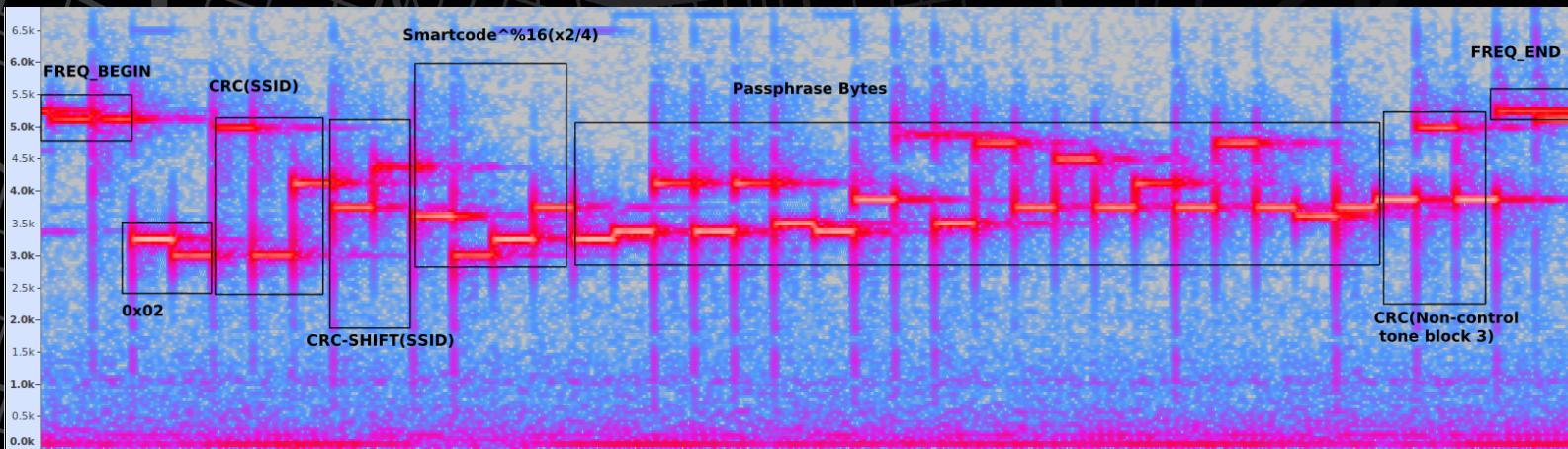


1. FREQ\_BEGIN
2. FREQ\_BEGIN
3. 0x12
4. SMARTCODE >> SHIFTED8
5. (SMARTCODE >> SHIFTED) >> SHIFTED8
6. CRC(0x12+(SMARTCODE>>SHIFTED8)+(SMARTCODE>>SHIFTED8>>SHIFTED8))
7. FREQ\_END
8. FREQ\_END

# Section 3

# Signal Structure

1. FREQ\_BEGIN
2. FREQ\_BEGIN
3. 0x02
4. CRC(SSID\_BYTES)
5. CRC-SHIFTED(SSID\_BYTES)
6. SMARTCODE%16 (x2/4)
7. PASSPHRASE\_BYTES
8. CRC(0x02+CRC(SSID\_BYTES)+CRC-SHIFTED(SSID\_BYTES) +SMARTCODE>>SHIFTED8+PASSPHRASE\_BYTES)
9. FREQ\_END
10. FREQ\_END



# 05

## Demo

Yes, we can reproduce the signal!

05

Demo

THAMYRIS: Replicate The Audio Protocol

# Limitations

1. Can't easily discover the camera's admin password
  - It's 6 HEX Characters
  - Password changes each time camera is reset, not tied to MAC or Serial#
2. Unable to decipher camera -> cloud communication
  - Camera has a local RSA keypair, that may change on each reset
  - Encrypted payloads are sent to the server
  - So, even though you can intercept the comm with a self-signed MITM, you can't see or make sense of the payload
3. You get what you pay for
  - Even when you know the password, it won't always connect (VLC is my 'go to')

# THANKS



7thzero.com



github.com/7thzero

# Technical References

- JADX: Dex to Java Decomplier - <https://github.com/skylot/jadx>
- Efficiency: Reverse Engineering with ghidra - <http://wapiflapi.github.io/2019/10/10/efficiency-reverse-engineering-with-ghidra.html>
- Guide to JNI (Java Native Interface) - <https://www.baeldung.com/jni>
- JDSP - Digital Signal Processing in Java - <https://psamabit9791.github.io/jDSP/transforms.html>
- Understanding FFT output - <https://stackoverflow.com/questions/6740545/understanding-fft-output>
- Spectral Selection and Editing - Audacity Manual -  
[https://manual.audacityteam.org/man/spectral\\_selection.html](https://manual.audacityteam.org/man/spectral_selection.html)
- Edit>Labelled Audio>everything greyed out - <https://forum.audacityteam.org/viewtopic.php?t=100856>
- Get a spectrum of frequencies from WAV/RIFF using linux command line -  
<https://stackoverflow.com/questions/21756237/get-a-spectrum-of-frequencies-from-wav-riff-using-linux-command-line>
- How to interpret output of FFT and extract frequency information -  
<https://stackoverflow.com/questions/21977748/how-to-interpret-output-of-fft-and-extract-frequency-information?rq=1>
- Calculate Frequency from sound input using FFT - <https://stackoverflow.com/questions/16060134/calculate-frequency-from-sound-input-using-fft?rq=1>
- Introduction - Window Size - <https://support.ircam.fr/docs/AudioSculpt/3.0/co/Window%20Size.html>
- Android: Sine Wave Generation - <https://stackoverflow.com/questions/11436472/android-sine-wave-generation>
- Android Generate tone of a specific frequency - <https://riptutorial.com/android/example/28432/generate-tone-of-a-specific-frequency>
- Android Tone Generator - <https://gist.github.com/slightfoot/6330866>
- Android: Audiotrack to play sine wave generates buzzing noise -  
<https://stackoverflow.com/questions/23174228/android-audiotrack-to-play-sine-wave-generates-buzzing-noise>

# Credits

CREDITS: This presentation template was created by [\*\*Slidesgo\*\*](#), including icons by [\*\*Flaticon\*\*](#), and infographics & images by [\*\*Freepik\*\*](#)