Android Reverse Engineering tools
Not the Usual Suspects

Axelle Apvrille - Fortinet
aapvrille@fortinet.com

Virus Bulletin, October 2017

Outline

Docker container with Android RE environment

You can **share** it with peers

Portable (Windows, Linux, Mac...)

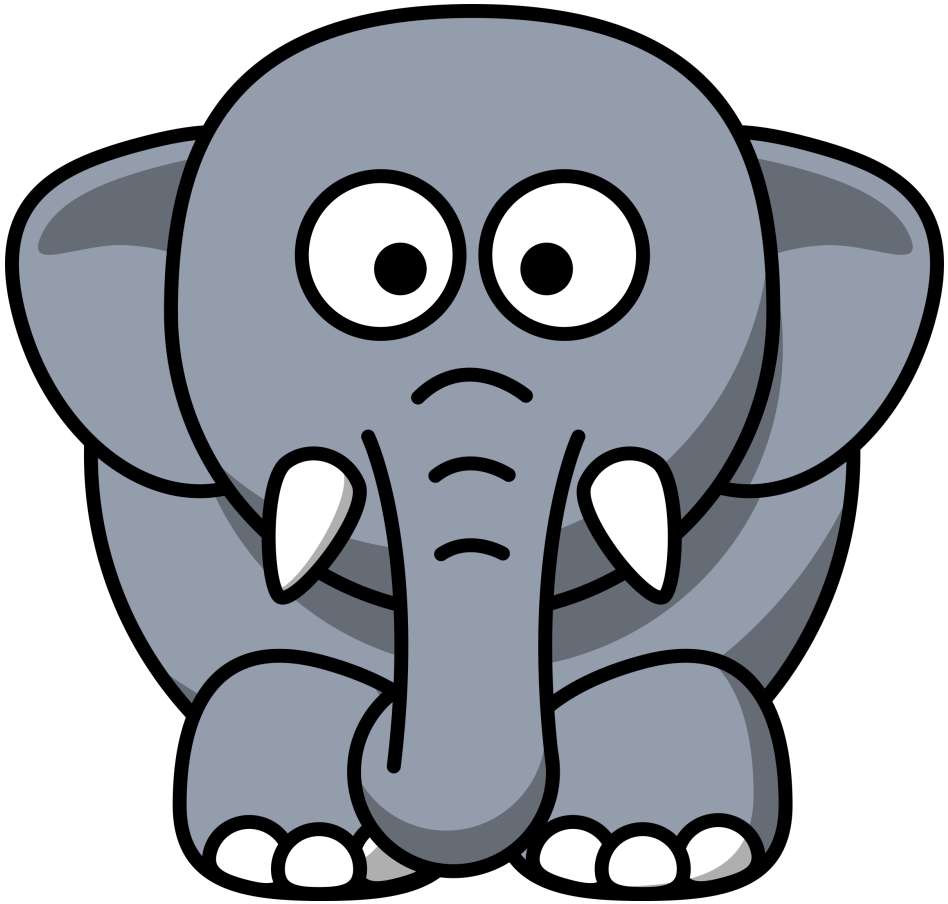
Install is as simple as:

```
docker pull cryptax/android-re
```

Docker container with Android RE environment

Download size: a few MB to 3 GB in worst cases

Docker container with Android RE environment



Lighter + better perf than a VM

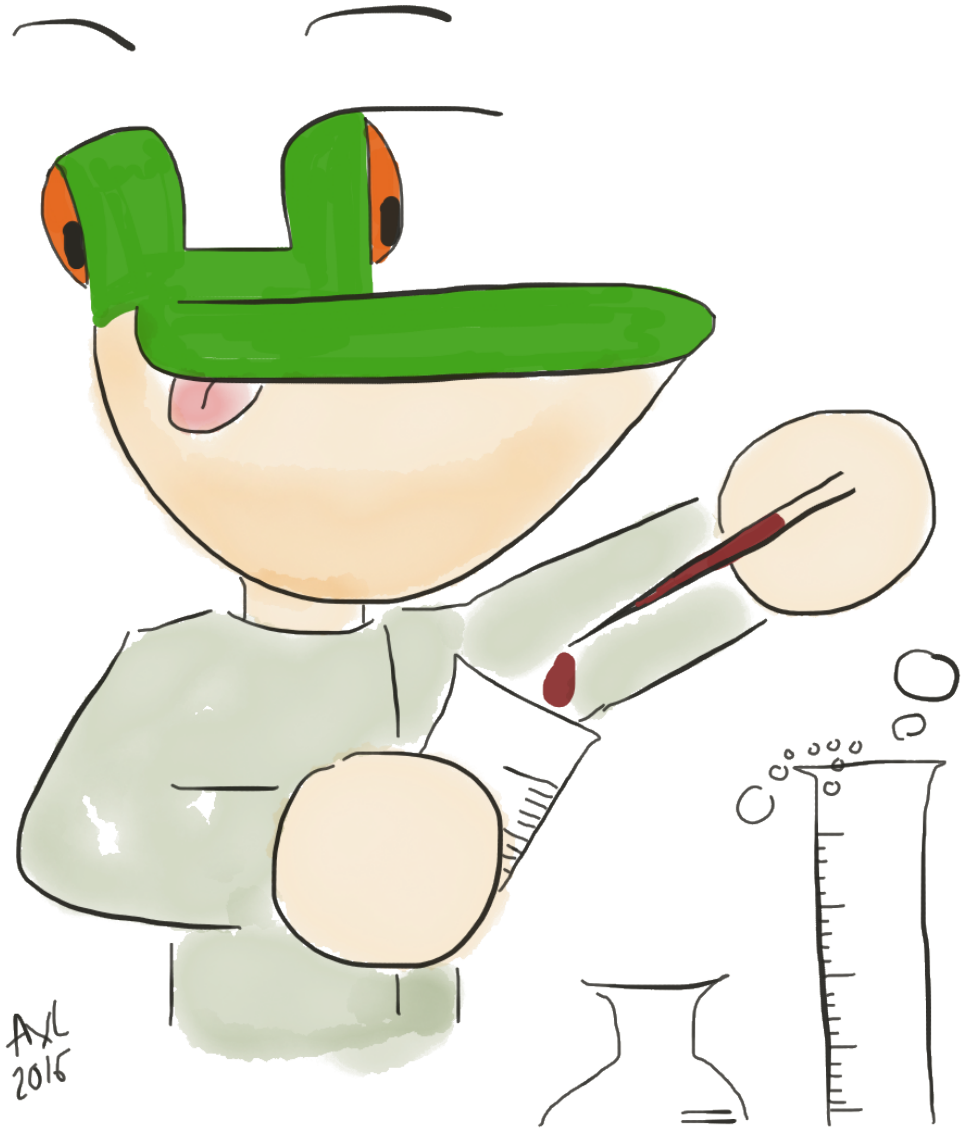
Download size with *VirtualBox*: **5 GB**

Docker container with Android RE environment

Open source: you can **customize**, enhance the container, easier to maintain

Dockerfile: `http://github.com/cryptax/androidre`

Demo



Launching several daemons in a container

Problem

```
CMD [ "command1" ]
CMD [ "command2" ]
```

Second command supersedes first one :(

Solution: Task Manager

- Install supervisor
- Configure `/etc/supervisor/conf.d/supervisord.conf` to launch both `cmd1` and `cmd2`
- `CMD [ "/usr/bin/supervisord" ]`

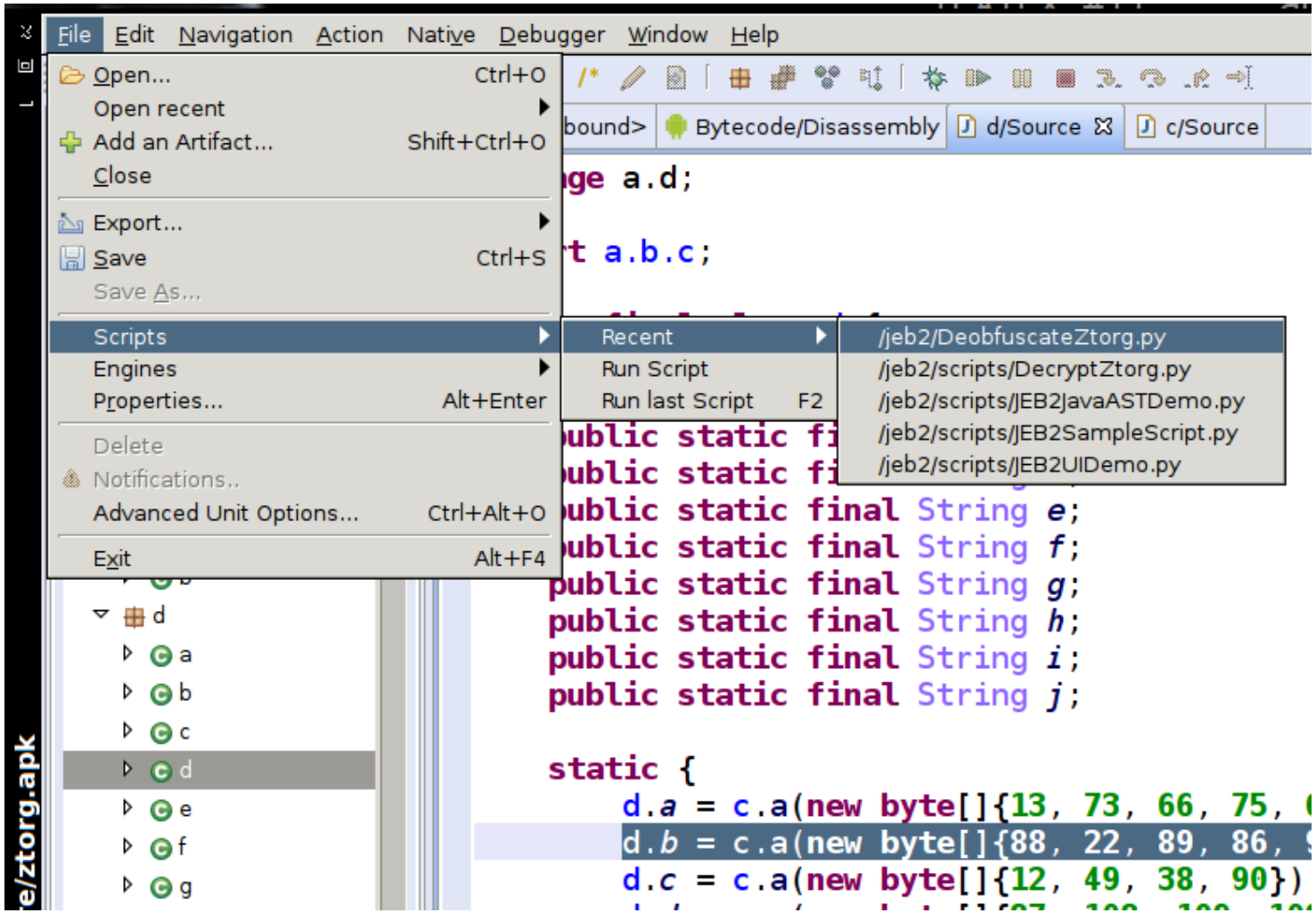
Installing the Android SDK

It can be scripted!

```
RUN wget -q -O "/opt/tools-linux.zip"
 ↪  https://dl.google.com/android/repository/tools_
 ↪  $ANDROID_SDK_VERSION-linux.zip
RUN unzip /opt/tools-linux.zip -d
 ↪  /opt/android-sdk-linux
RUN echo y | android update sdk --filter tools
 ↪  --no-ui --force -a
RUN echo y | android update sdk --filter
 ↪  platform-tools --no-ui --force -a
...
```

Outline

JEB2 scripts: automating reverse engineering tasks



Note: I am not affiliated to PNF software

Case study: De-obfuscate Android/Ztorg strings

Android/Ztorg is an active family of advanced Android trojan:

- Anti-emulator features
- String obfuscation
- Downloads remote content

Our goal: de-obfuscate strings

```
d.a = c.a(new byte[]{13, 73, 66, ...});
```

```
d.a = "channel/channel.txt";
```

Get inspiration from existing scripts

```
$ cd ./scripts
$ ls
JEB2AsyncTask.py
JEB2JavaASTDecryptStrings.py
JEB2JavaASTDemo.py
...
```

Open and edit JEB2JavaASTDecryptStrings.py

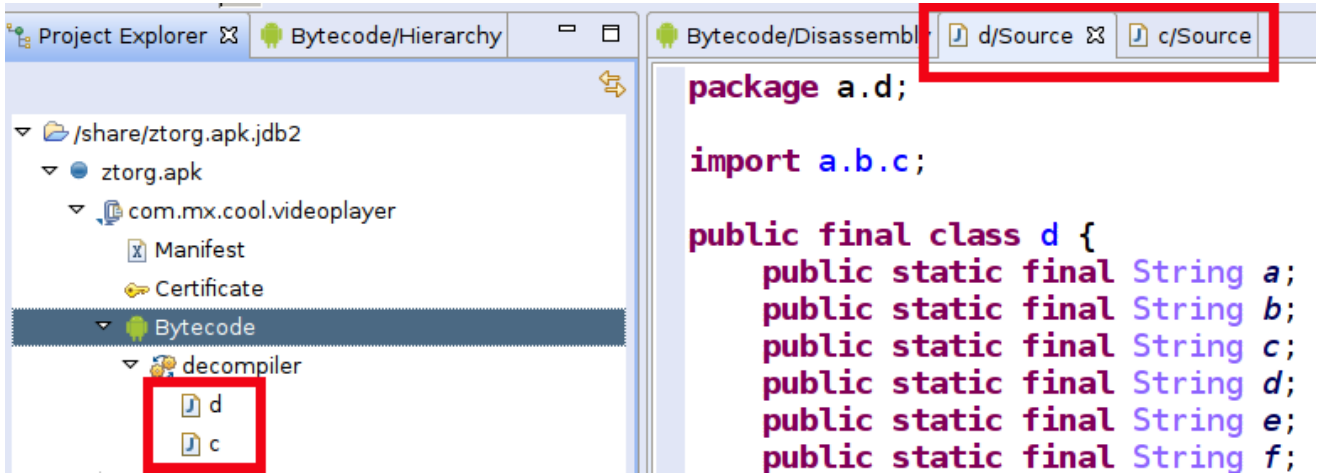
Resources: https://github.com/pnfsoftware/jeb2-samplecode/tree/master/scripts

Get first opened project = sample

```
class JEB2JavaASTDecryptStrings(IScript):

  def run(self, ctx):
    engctx = ctx.getEnginesContext()
    if not engctx:
      print('Back-end engines not initialized')
      return

    projects = engctx.getProjects() # get all
     ↪  opened projects
    if not projects:
      print('There is no opened project')
      return

    prj = projects[0] # get first project
```

Get decompiled code units = decompiled class

Our script will process all decompiled sources we have opened.




```
self.codeUnit =
↪  RuntimeProjectUtil.findUnitsByType(prj,
↪  ICodeUnit, False)[0] # bytecode disassembly
self.units =
↪  RuntimeProjectUtil.findUnitsByType(prj,
↪  IJavaSourceUnit, False) # Java source
```

Remove code specific to Android/Obad

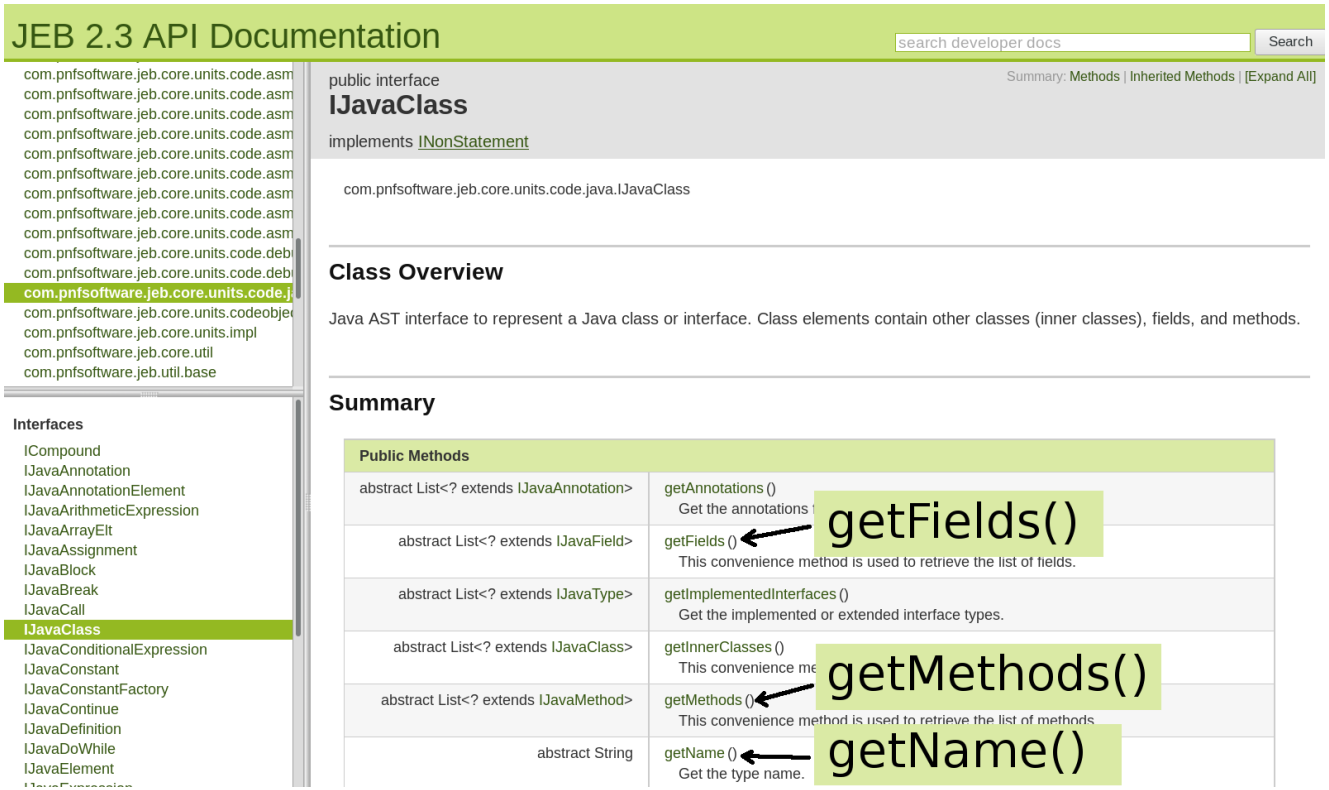
Remove this: completely different for Android/Ztorg!

```
 if not projects:
     print('There is no opened project')
     return

   prj = projects[0]
...
# the encryption keys could be determined by
 ↪  analyzing the decryption method
self.targetClass = 'MainActivity'
self.keys = [409, 62, -8]
...
units = RuntimeProjectUtil.findUnitsByType(prj,
 ↪  IJavaSourceUnit, False)
```

Get class object

```
for unit in self.units: # for each decompiled source
    javaClass = unit.getClassElement()  # get class
```

The type of `javaClass` is **IJavaClass**



Locate static constructor

In Android/Ztorg, obfuscated strings are grouped in the **static constructor**.
Let's locate the static constructor of our class.

```
for m in javaClass.getMethods():
  if m.getName() == '<clinit>': # only in static
  ↪  constructors
```

Locate an assignment

Methods (and constructors) are made of *statements* (lines).

```
    value = c.a(...);
```

We are looking for a **assignment**.
Resource: List of statement types

```
for statement in m.getBody(): # read all lines
  if statement.getElementType() ==
   ↪  JavaElementType.Assignment:
```

Locating calls to de-obfuscating routine

```
d.a = c.a( byte array );
```

- **left** : the variable `d.a`
- **right** : what we assign
- In our case, we are interested in lines with a **call** to our de-obfuscating routine `c.a()`

```
decode_method = 'La/b/c;->a([B)Ljava/lang/String;'
 ↪  # prototype of deobfuscation routine
  if isinstance( statement.getRight() ,IJavaCall)
   ↪  and  statement.getRight() ⌋
   ↪  .getMethod().getSignature() ==
   ↪  decode_method}:
```

Retrieve the obfuscated bytes

1. Get the **arguments** of our call
2. Is it a `new byte [] ...` ?

```
d.a = c.a(new byte[]{13, 73, 66, 75, 6...});
```

3. If so, get the values and store them in a Python array (encbytes)

```
for argument in elem.getArguments():
  if isinstance(argument, IJavaNewArray):
    encbytes = []
    for v in argument.getInitialValues():
      # retrieve the encoded values
      encbytes.append(v.getByte())
```

De-obfuscate the bytes

Implement the routine in Python, using reverse engineering of sample

```
def decodeBytes(self, buf):
    key0 = buf[0]
    key1 = buf[len(buf)-1]

    # copy buffer
    result = buf[1:len(buf)-1]

    # decode
    for i in range(0, len(result)):
      result[i] = result[i] ^ key1
      result[i] = result[i] ^ key0

    return result
```

Modify the line and replace with de-obfuscated string

- **replaceSubElement** replaces part of a statement

```
replaceSubElement( oldElement , newElement )
```

- **oldElement** is `c.a(new byte [] {...})`
- **newElement** is the deobfuscated string
- Convert byte [] to string: `''.join(map(chr, decbytes))`

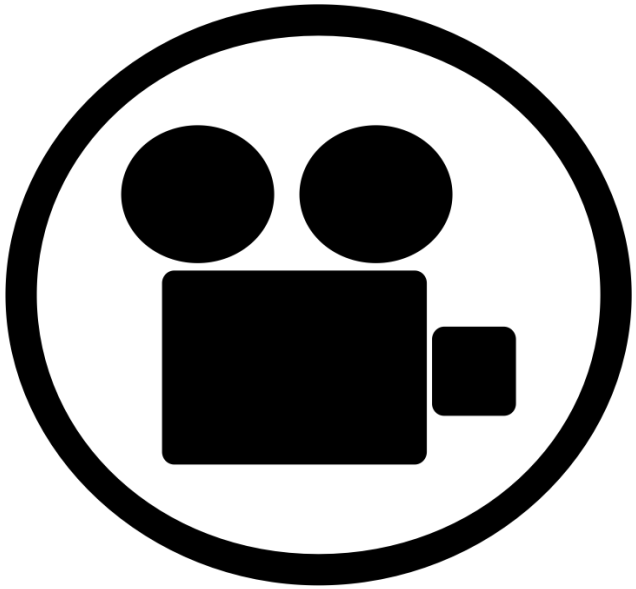
```
decbytes = self.decodeBytes(encbytes)
deobfuscated_string = self.cstbuilder.createString(
↪  (''.join(map(chr,decbytes)))
father.replaceSubElement( elem ,
↪  deobfuscated_string )
```

Notify / Refresh the UI

```
unit.notifyListeners(JebEvent(J.UnitChange))
```

DONE - JEB2 script is finished

Have a look



As simple as loading the script and **so helpful**

http://github.com/cryptax/misc-code

Outline

Running a sample step by step

- Rather heavy
- Launches an Android emulator
- Recompiles the sample (check corporate ethics)
- Has **improved much since March 2017**

JEB2

You can also jump into native **ARM** code!

https://www.pnfsoftware.com

CodeInspect

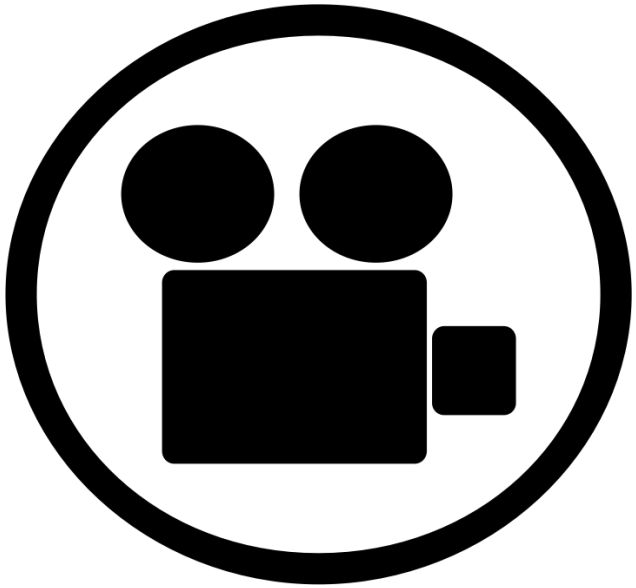
It's not **smali**, it's not **Java**, it's ... **Jimple**!

https://codeinspect.sit.fraunhofer.de/

Step debugging with CodeInspect

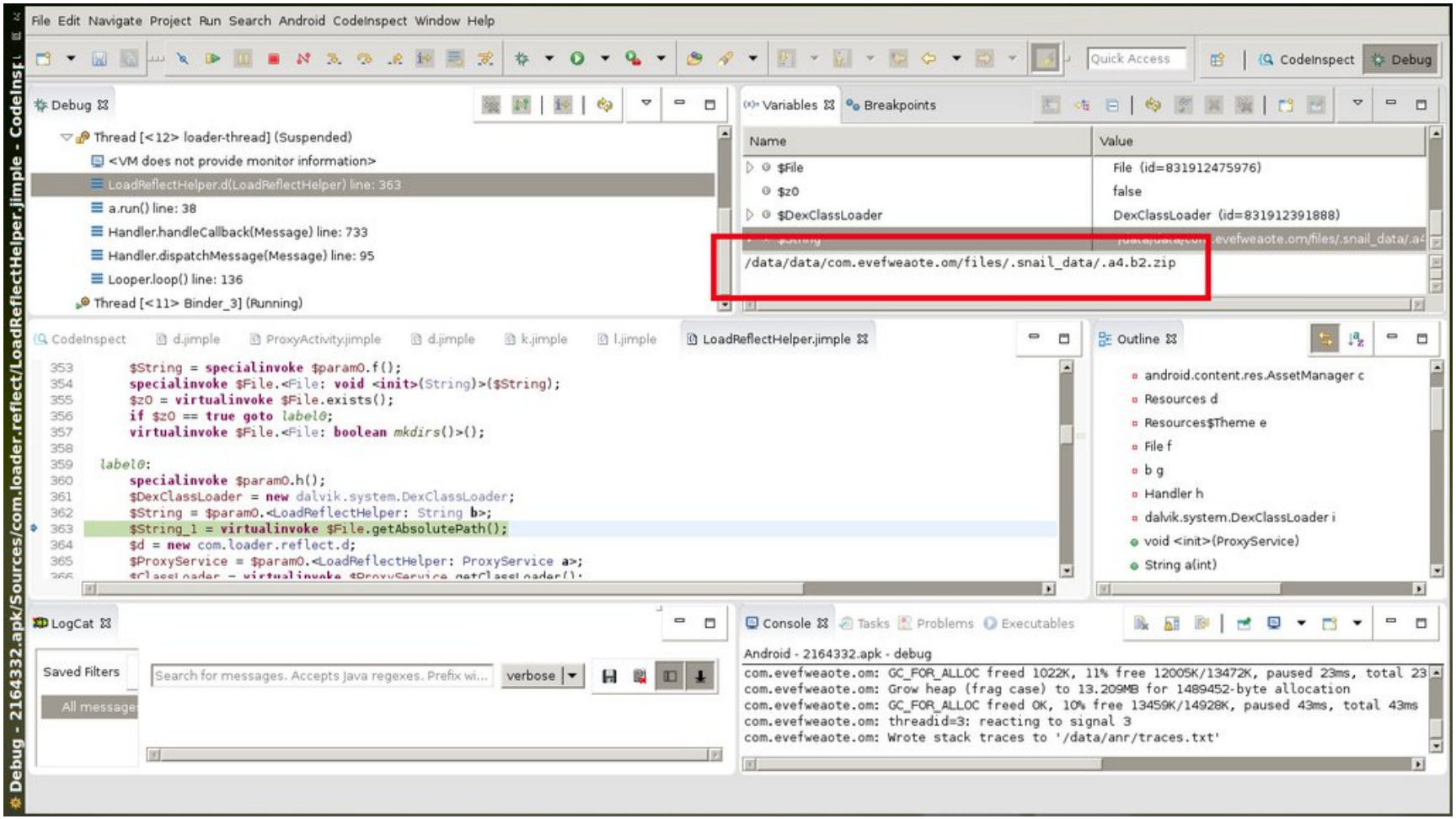
Problem: Riskware/InnerSnail!Android loads a DEX file, but it's **difficult to find its name** with static analysis.

Solution: **step debug** the riskware



Note: I am not affiliated to CodeInspect

Step debugging with CodeInspect (backup slide)

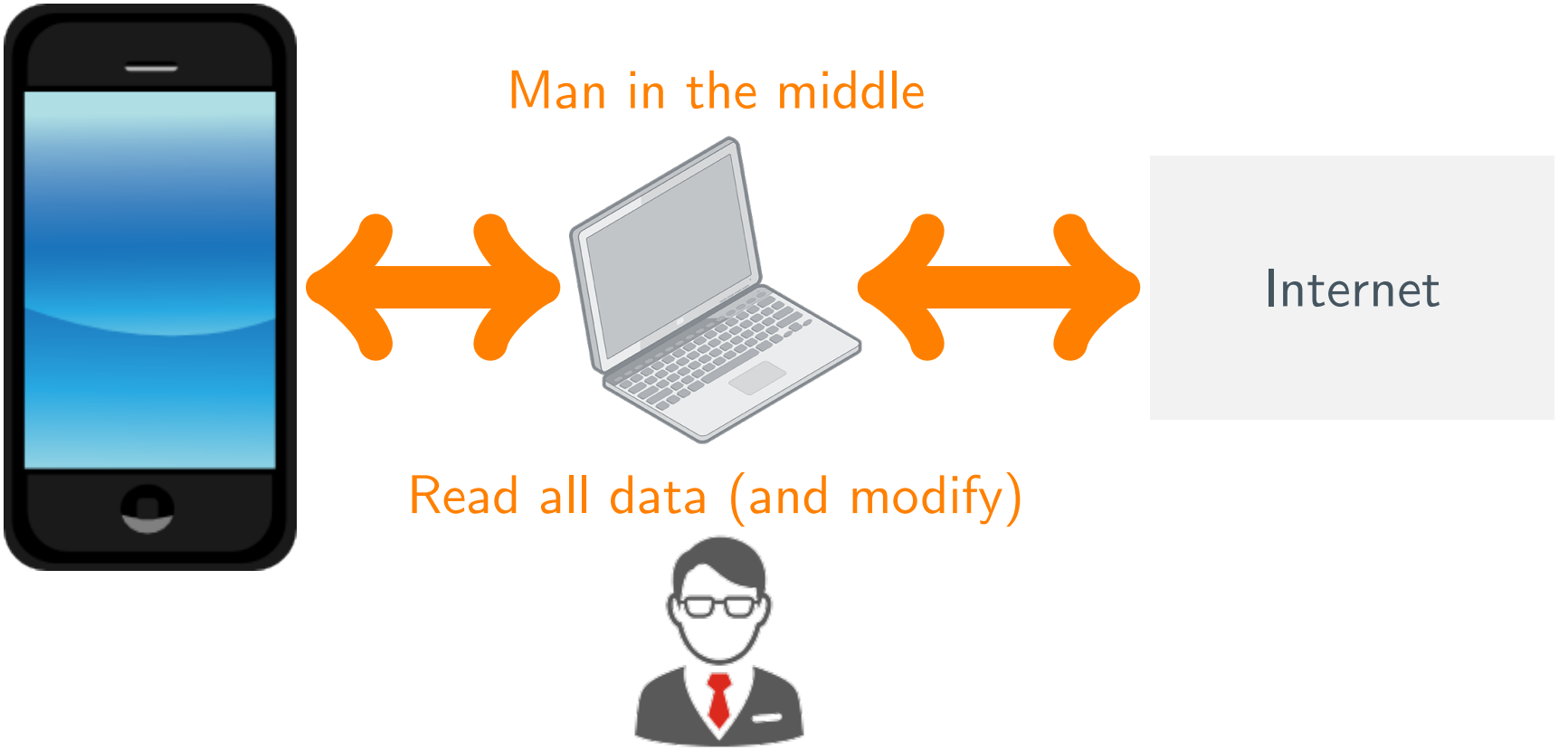


Outline

HTTPS Flow inspection

Internet

HTTPS Flow inspection



HTTPS Flow inspection

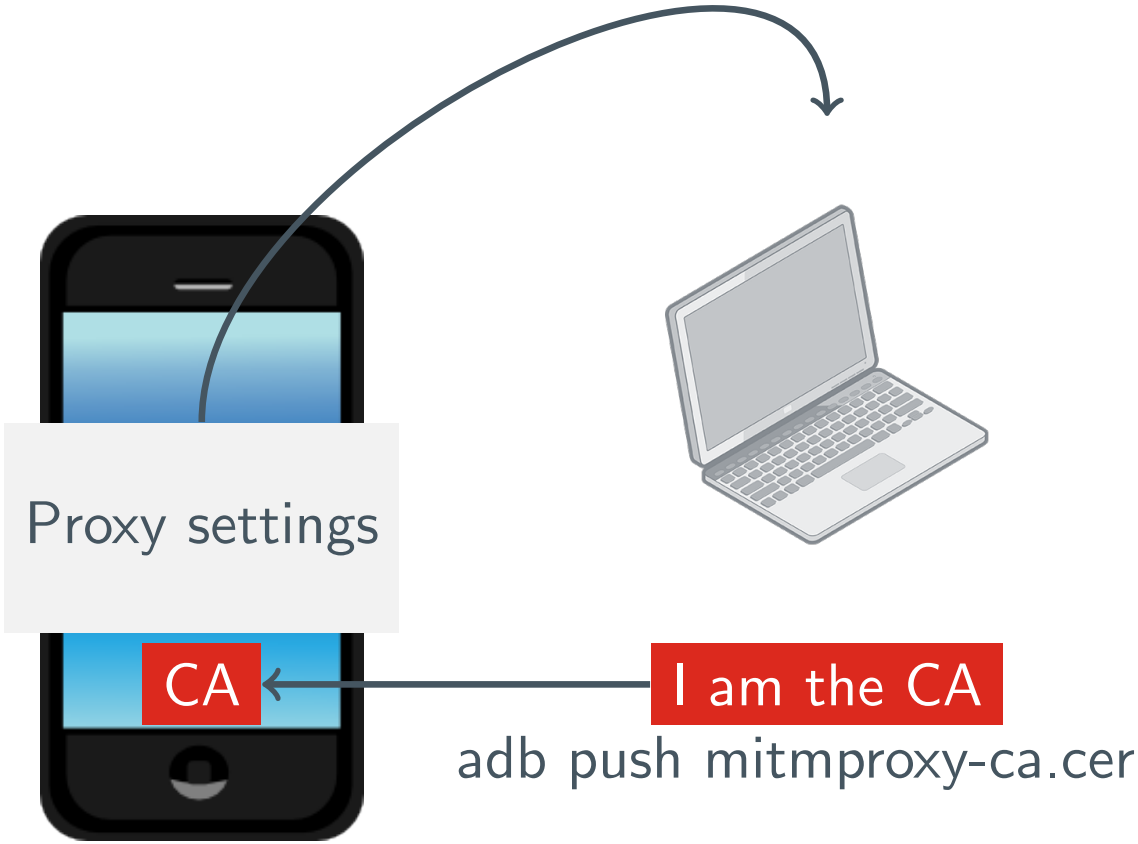
Man in the middle

Internet

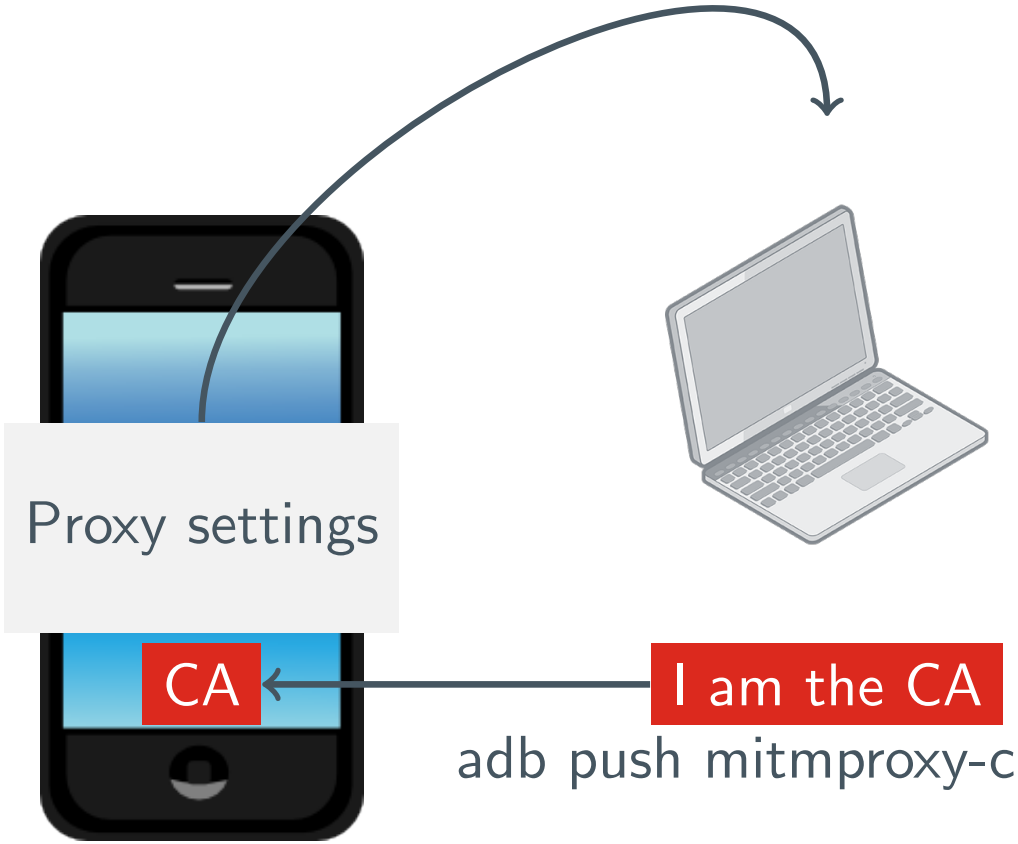
Read all data (and modify)

Tool: mitmproxy

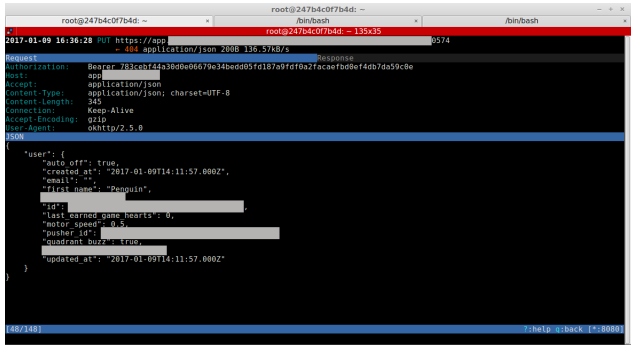
HTTPS Flow inspection



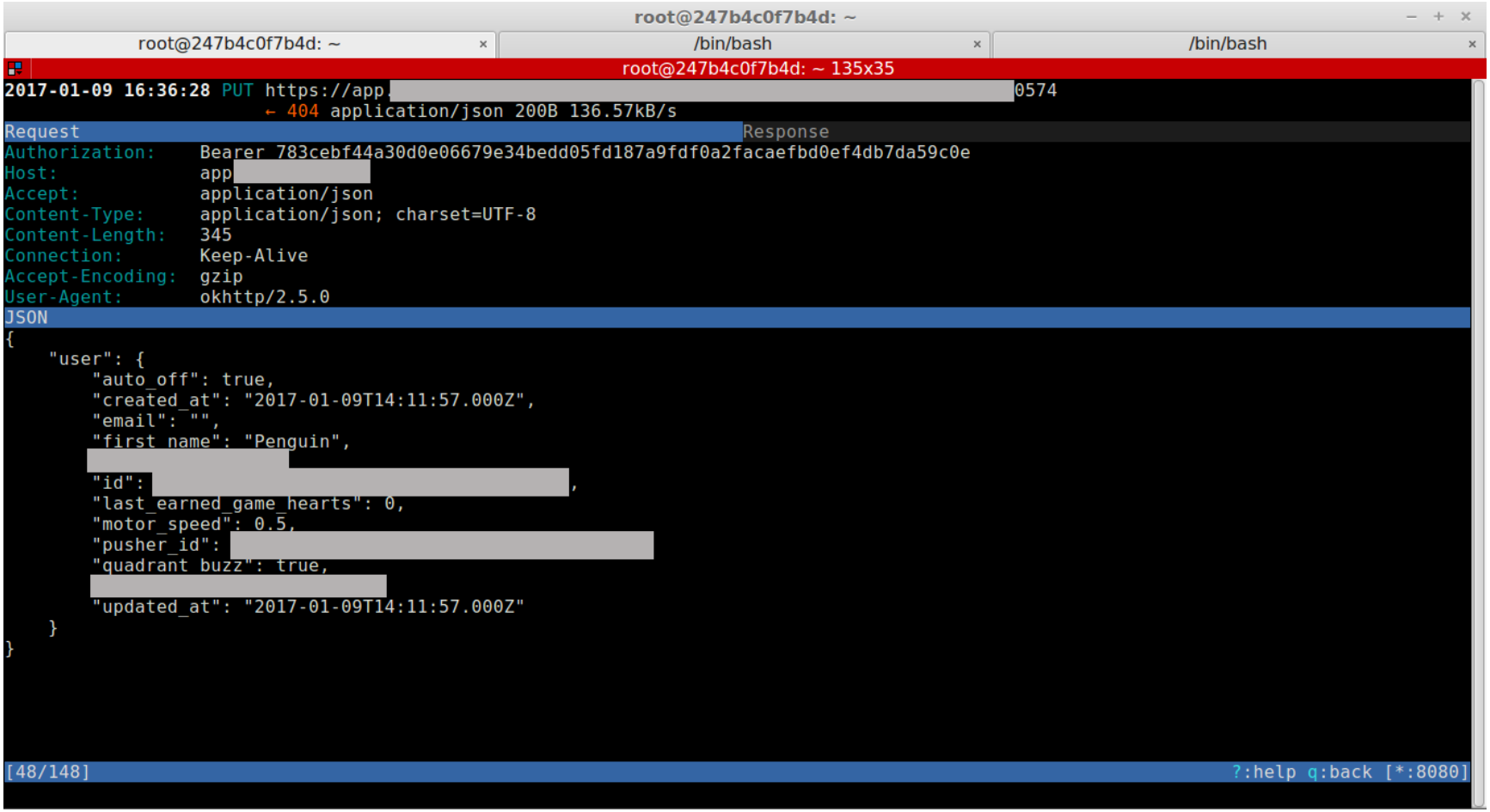
HTTPS Flow inspection



$ ./mitmproxy

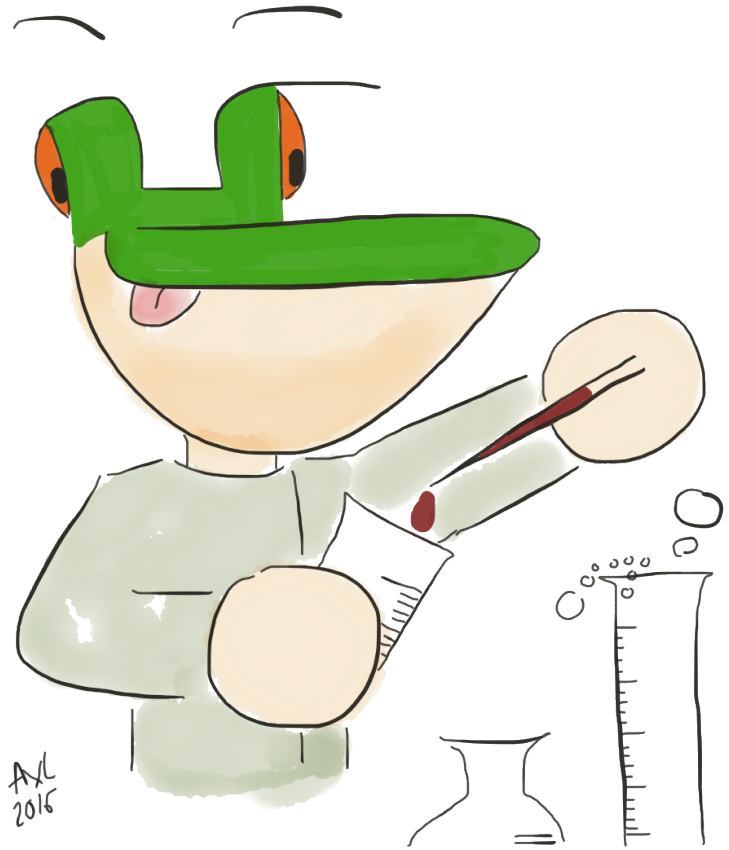


Mitmproxy: example on Android



Outline

Demo



Radare2 de-obfuscating script on Android/Ztorg

http://github.com/cryptax/misc-code

Radare2 for Dalvik: take away

Shortest cheat sheet ever ;-)

- Launch: `r2 classes.dex`
- Searching: iz~mystring, ic~mystring, afl~mystring
- Cross references to: **axt** name, from: **axf** name
- Comment: CC mycomment

R2 scripts

- In the script:

```
import r2pipe
r2p = r2pipe.open()
r2p.cmd('s 0xdeadbeef') # launch a R2 command
```

- Launching the script: `#!pipe python file.py args`

Thanks for your attention!

Questions?

Shameless ad

Smart devices CTF (including Android)

Nov 29 - French riviera

https://ph0wn.org