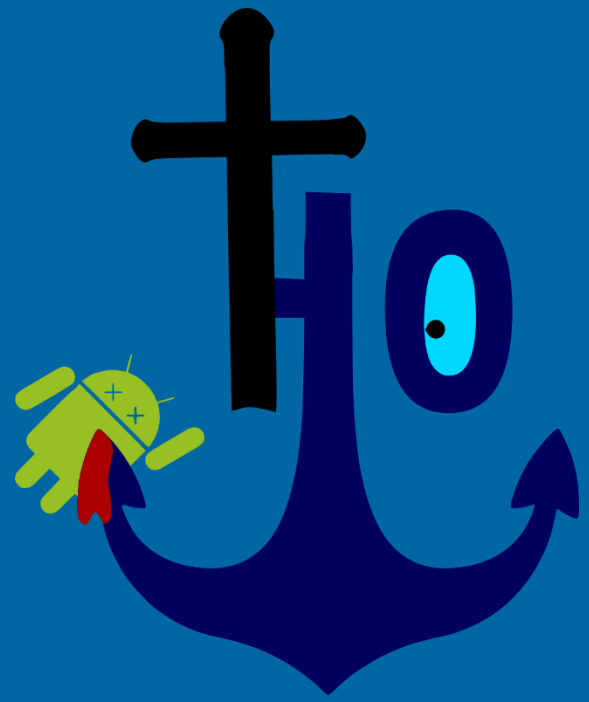


Hook as you want it

Digital Security

Digital Security

WHO AM I ?

Benoît FORGETTE
IT Security Engineer, IOT expert @Digital.Security
@Mad5quirrel

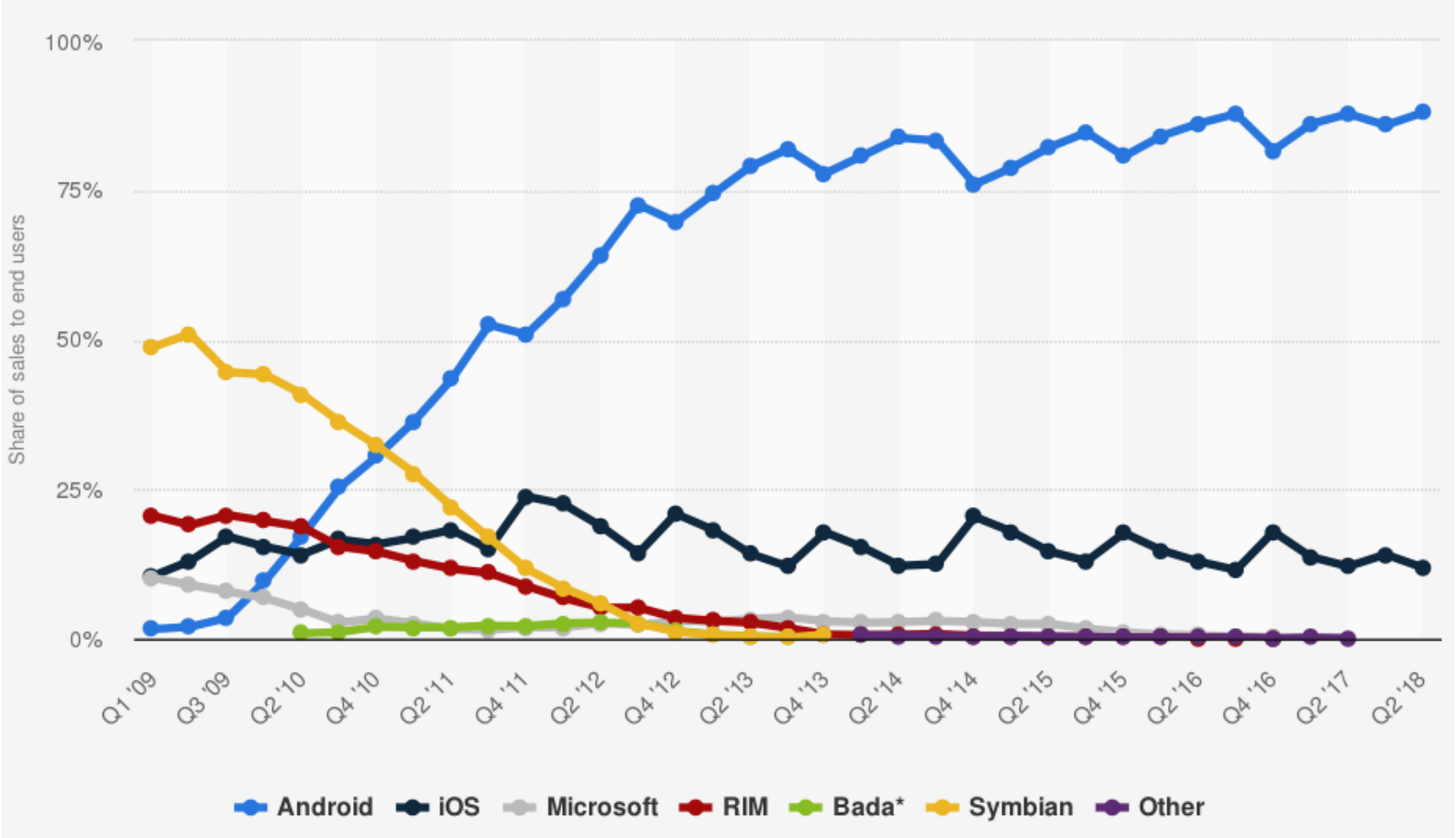
OUTLINE

1. Why Android?
2. Existing tools
3. What is ASTHOOK ?
4. What is the execution routine ?
    1. Manifest
    2. Source code
    3. Dynamic analysis
5. What is next ?

1. Why Android ?

Digital Security

1. Smartphone's market



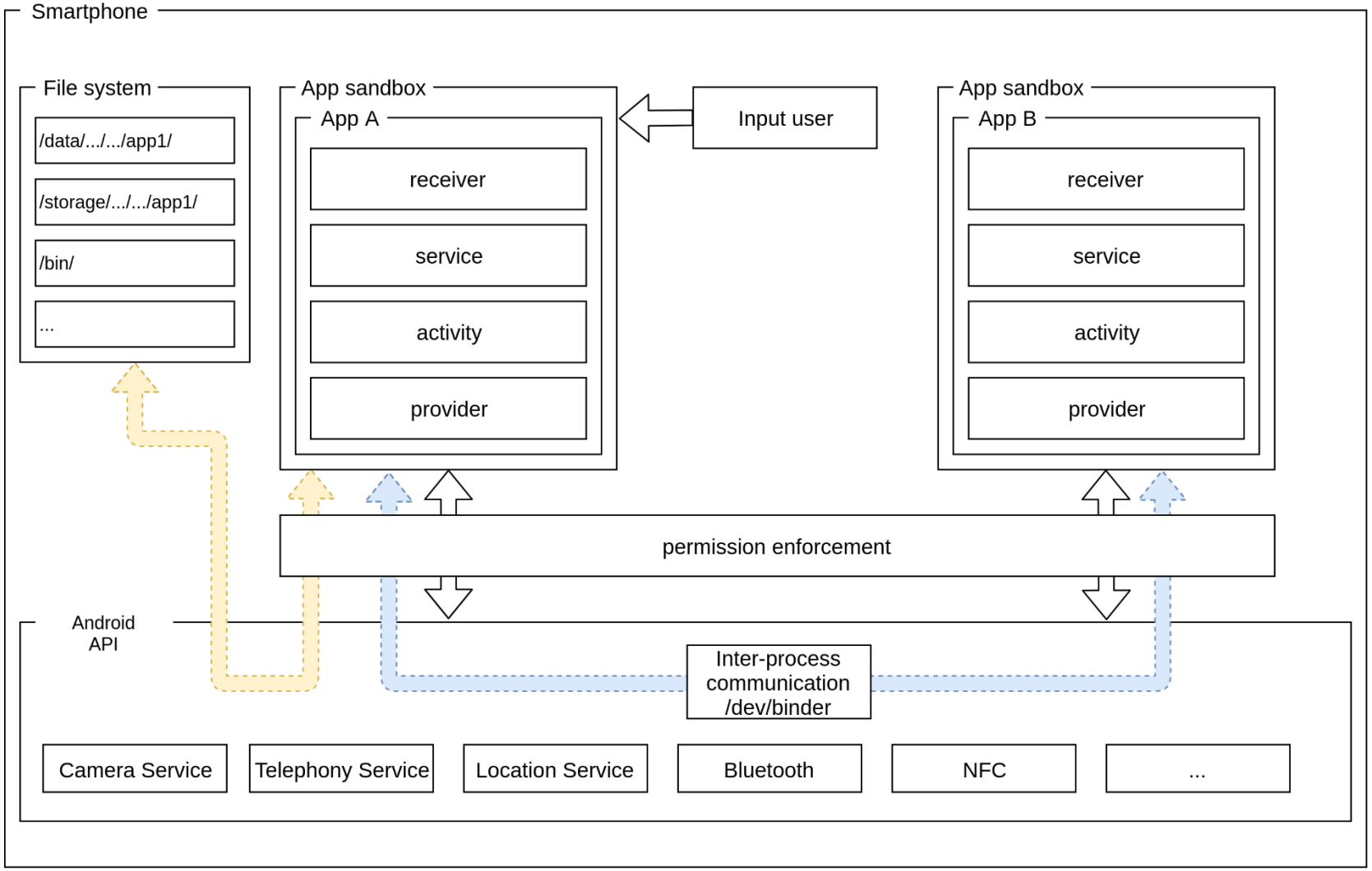
1. Why Android rules the world ?

- Standard Bootloader
- API for hardware
- Arm trustzone
- High level language
- IPC (Inter process communication)

Digital Security

1. Why Android rules the world ?

Android Functionality mechanism



Digital Security

2. Existing tools

Digital Security

2. Existing tools

	MobSF	Androguard	Quark	Frida	Drozer
Static Analyis	Yes	Yes	Yes	No	Partial
Dynamic Analysis	Yes	No	No	Yes	Yes
Plugins options	No	No	Yes	Snippets	Yes
AST analysis	Partial	Yes	Yes	No	No
Android Binary XML Format readder	Yes	Yes	No	No	No
Generate APK to exploit vulnerability	No	No	Yes	No	No
Perform Sql Injection	No	No	No	No	Yes
Auto-decompilation	Yes	Yes	Yes	No	No
Dynamic Hook	No	No	No	Yes	No
Manipulate process memory	No	No	No	Yes	Partial

3. What is ASTHOOK ?

Digital Security

3. Why Asthook ?

Reuse

- Modular plugins
- AST analysis
- Taint Analysis
- POC generation
- Can be used as emulator or on physical device.
- Can be used as standalone
- With root or not rooted phone

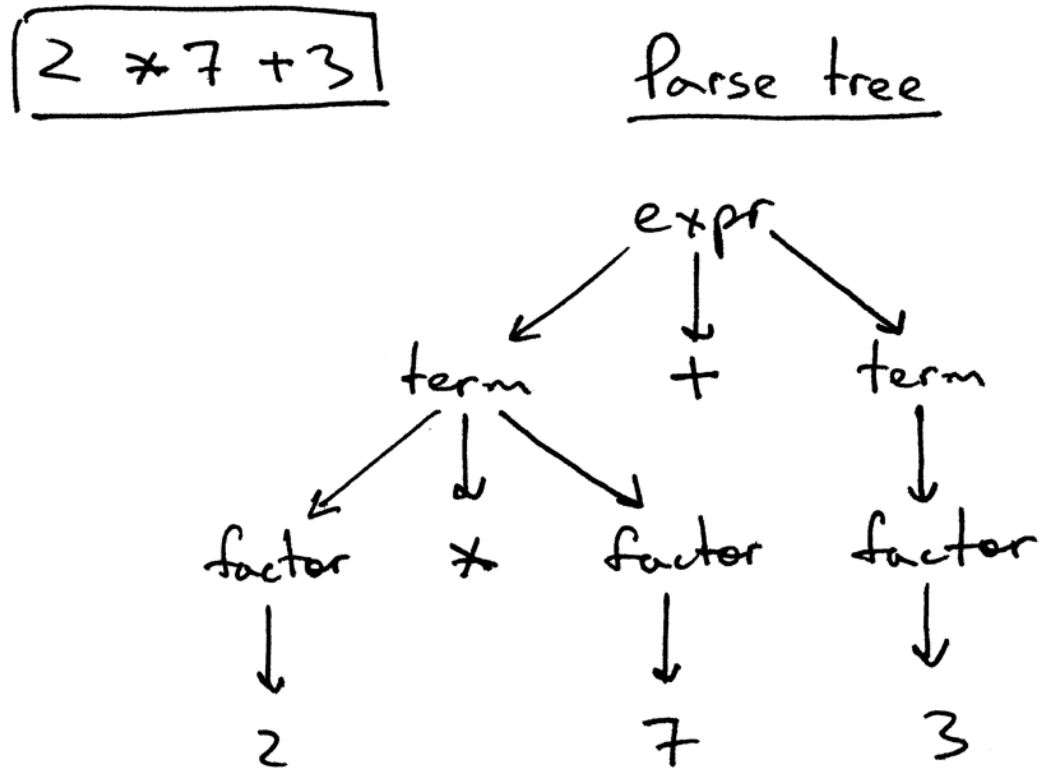
New

- Dynamic analysis can use static analysis result
- **Instrumentation of the phone**

Digital Security

3. StoryTime

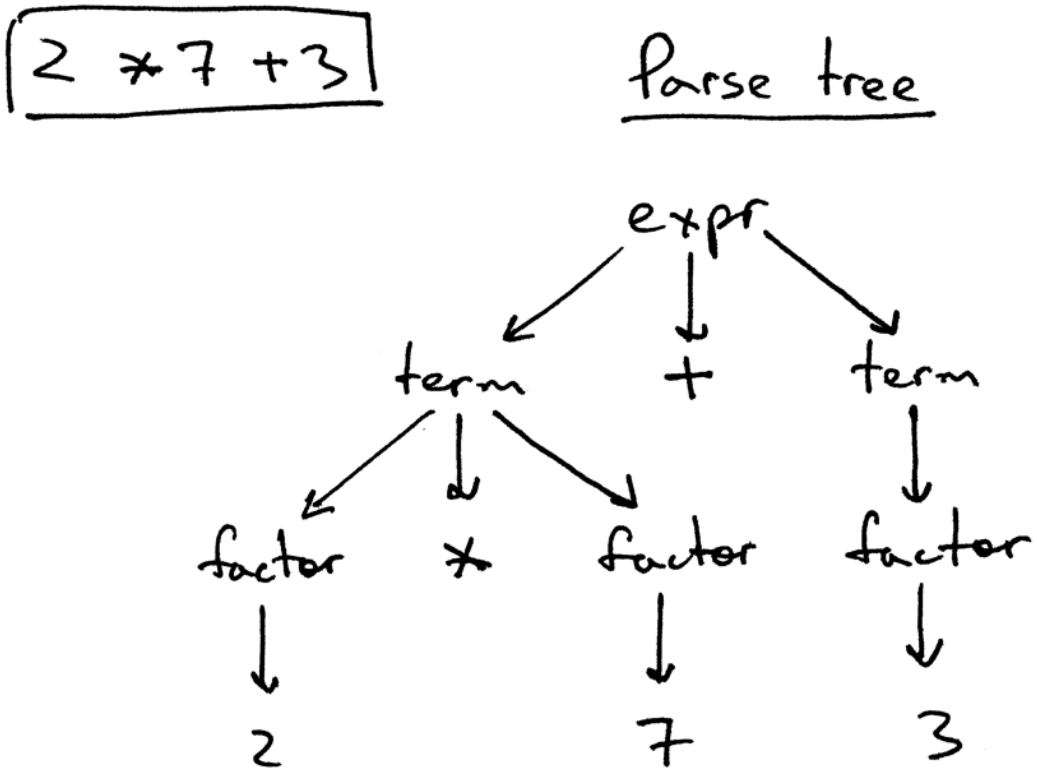
AST



Digital Security

3. StoryTime

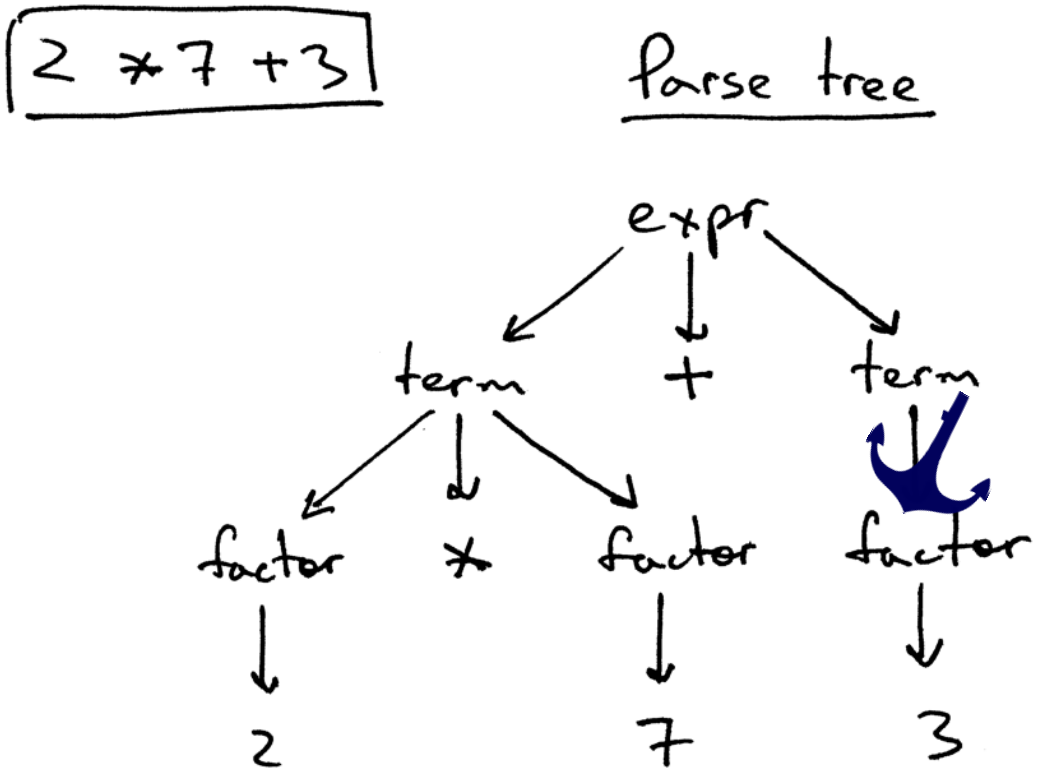
AST HOOK



Digital Security

3. StoryTime

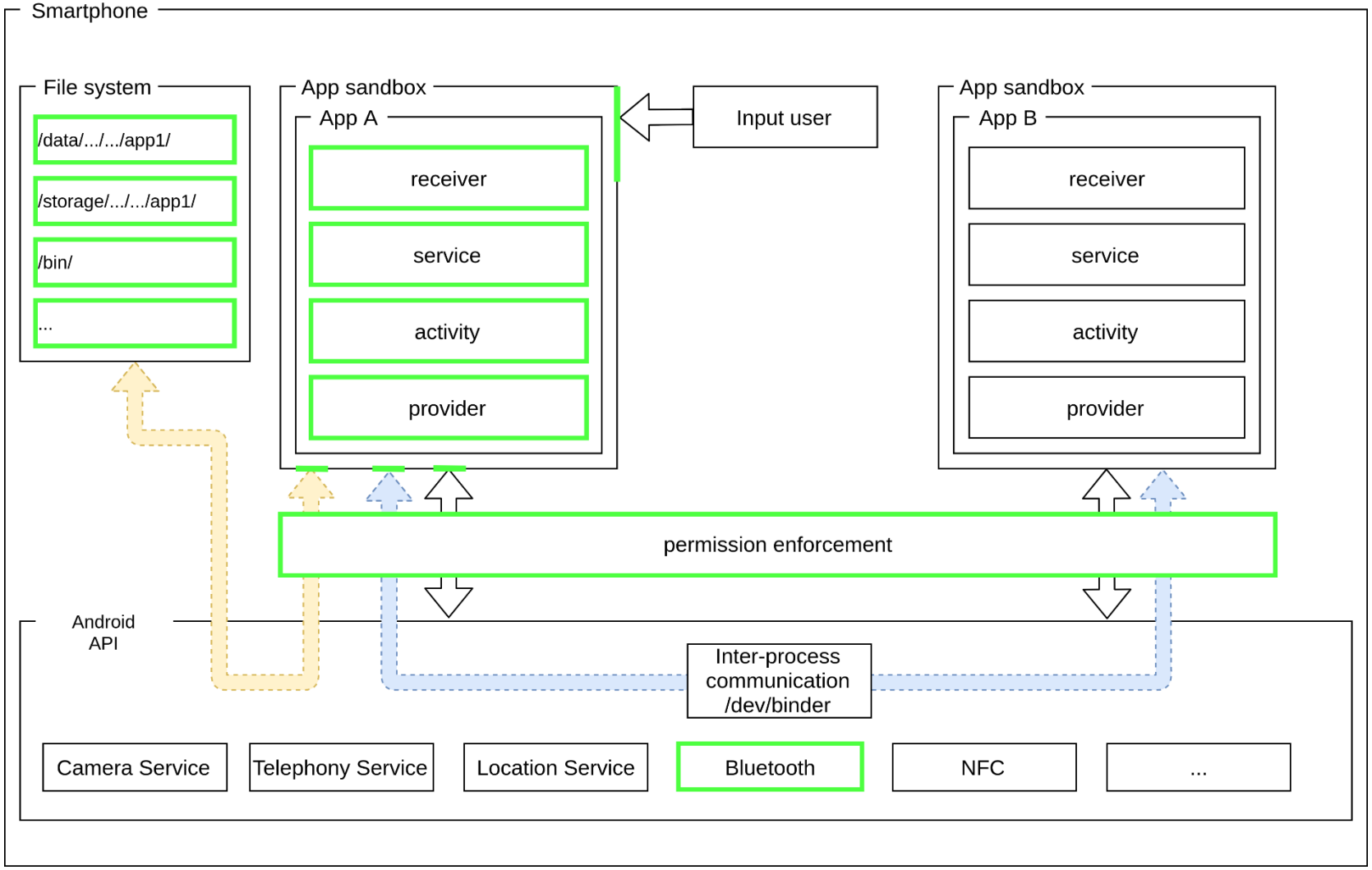
AST HOOK



Digital Security

3. What does ASThook analyze ?

Android Functionality mechanism



4. What is the execution routine ?

Digital Security

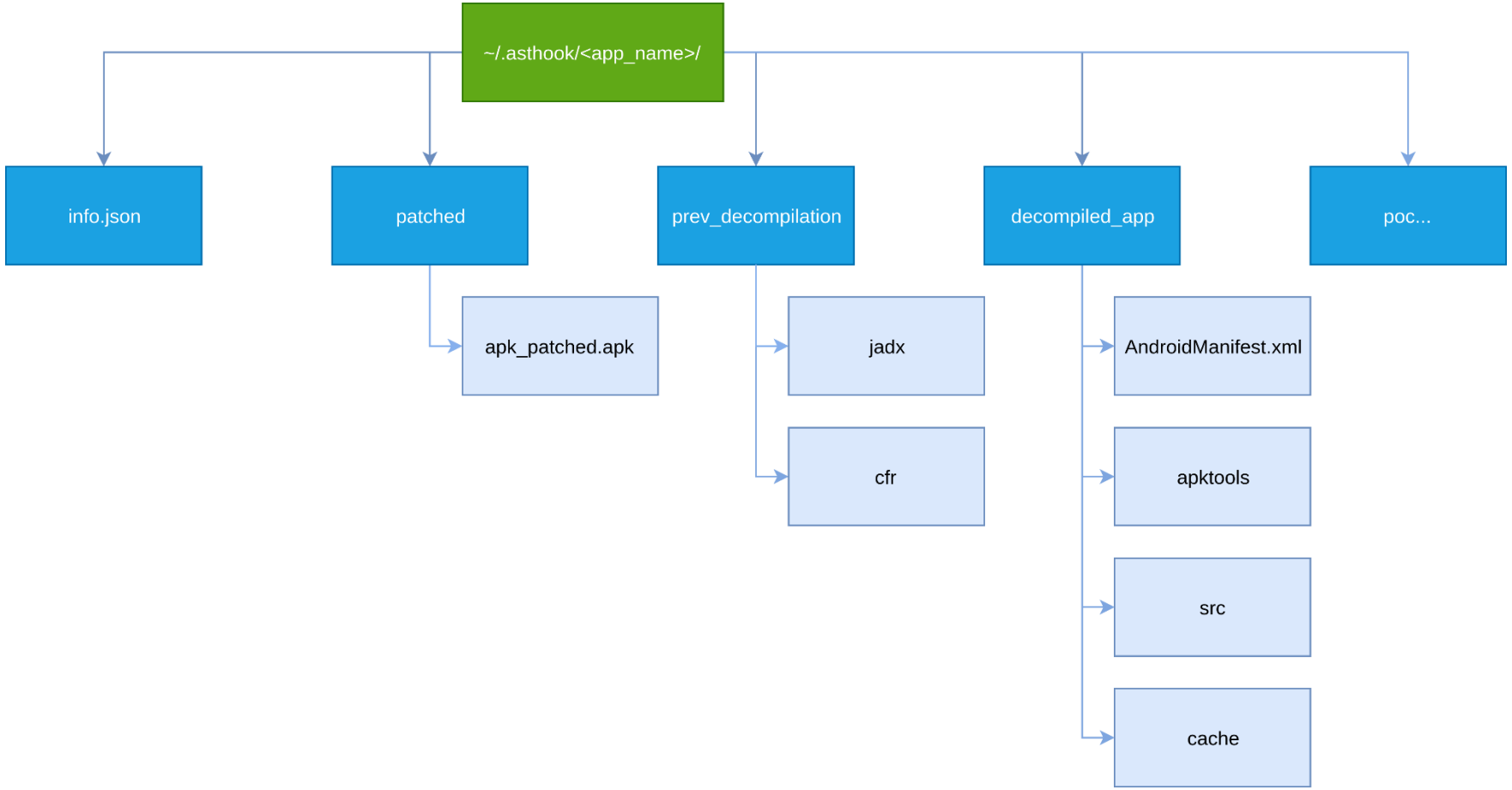
4. Decompilation

- **ASTHOOK** can use these decompilers:
  - cfr
  - Procyon
  - Jadx
  - Jd-core
  - Fernflower

```
python3 ./src/asthook.py misc/sieve.apk --decompiler cfr
Extracting misc/sieve.apk to sieve
Converting: classes.dex -> classes.jar (dex2jar)
Decompiling to sieve/src (cfr)
I: Using Apktool 2.3.4-dirty on sieve.apk
I: Loading resource table...
I: Decoding AndroidManifest.xml with resources...
I: Loading resource table from file: /home/bfo/.local/share/apktool/framework/1.apk
I: Regular manifest package...
I: Decoding file-resources...
I: Decoding values */* XMLs...
I: Baksmaling classes.dex...
I: Copying assets and libs...
I: Copying unknown files...
I: Copying original files...

(C) Copyright 2020 Benoit FORGETTE
Author: Benoit Forgette `MadSquirrel` <benoit.forgette@ci-yow.com>
This program is free software; you can redistribute it and/or
modify it under the terms of the GNU General Public License
as published by the Free Software Foundation; version 3
of the License.


################################################################################
                                    Static Analysis
################################################################################

com.mwr.example.sieve
Build: ? latest ?

************************** Dangerous functionnality **************************
```

Digital Security

4. Structure of the project



4.1. Manifest

Digital Security

4.1 Overview of the static analysis

When the tool is launched without option, only a quick analysis of the Manifest is executed.
The tool enumerates:

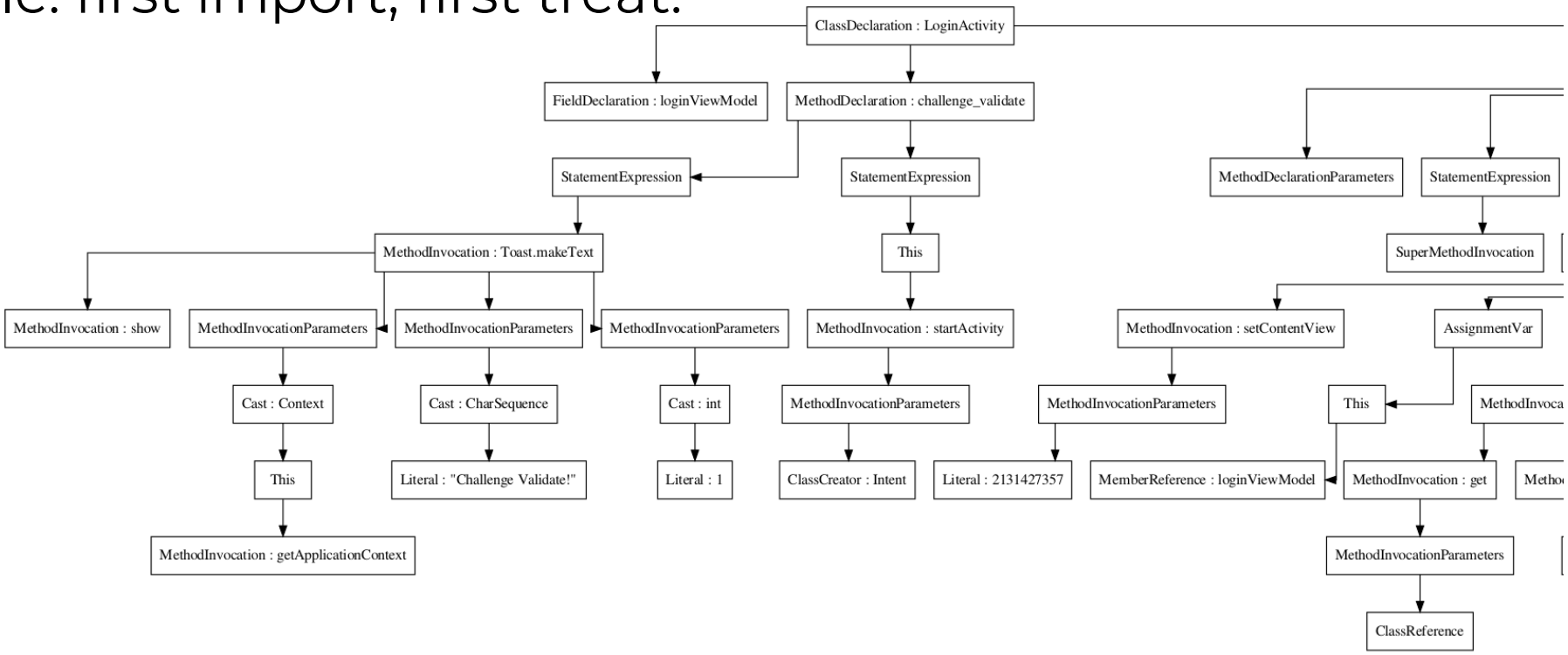
- Dangerous functionalities
- Permissions used and created
- Activities exported
- Services exported
- Broadcast receivers exported
- Providers

4.2 Source code

Digital Security

4.2 AST build option

- --tree option builds an Abstract Syntactical Tree of the source code.
- The order used to build the tree is by analyzing all Java files with the rule: first import, first treat.



Digital Security

4.2 Plugin as a hook

- Each plugin can hook any node of the AST and get informations of the node and execute code.

File: **asthook/static/module/ListUserInput/ListUserInput_node.py**

```
1  from asthook.static.ast import Node
2  from asthook.utils import *
3
4  @Node("LocalVariableDeclaration", "in")
5  class LocalVariableDeclaration:
6      @classmethod
7      def call(cls, r, self):
8          if self.elt.type.name == "EditText" or \
9                  self.elt.type.name == "TextView":
10                 Output.add_tree_mod("user_input",
11                                     "EditText/TextView",
12                                     ["%s" % self.elt.declarators[0].name,
13                                      r["Filename"],
14                                      self.elt._position], r["instance"])
15         return r
```

Digital Security

4.2 Plugin interface for static analysis

Digital Security

4.2 Taint analysis

- --taint option enables the taint analysis

Definition:
Taint analysis allows to follow a variable on each use in the source code.
This taint is done on object level.

Digital Security

4.2 Taint analysis

To create a taint analysis, the lifespan of a variable should be determined.

To do that we identify each scope in the source code.

Definition:
A scope is a zone where each variable defined inside will be destroyed when the scope ends:

- body of a function
- Parameter of a for loop
- Body of a class
- Body of a namespace
- Etc.

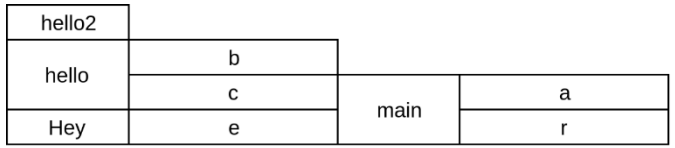
Digital Security

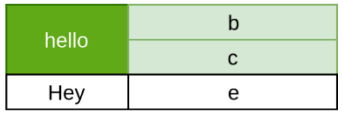
4.2 Taint analysis

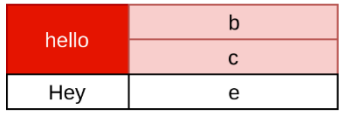
For instance:

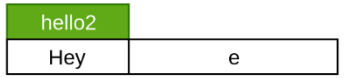
```
class Hey:

  e = "hello"
  def hello(b, c):
    print(f"{e}:{b} {c}"

  def hello2():
    print(e)

def main(a):
  r = "test"
  Hey.hello(a, r)
```

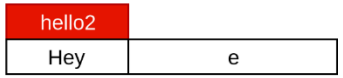
Scopes creation:

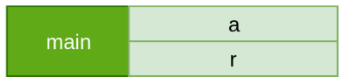


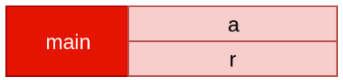












Digital Security

4.2 Taint analysis

Binding of variables

For each use of the variable, a link needs to be done with its previous use.

For instance:

```
class Hey:

  e = "hello"
  def hello(b, c):
    print(f"{e}:{b} {c}"

  def hello2():
    print(e)

def main(a):
  r = "test"
  Hey.hello(a, r)
```

Digital Security

4.2 Taint analysis

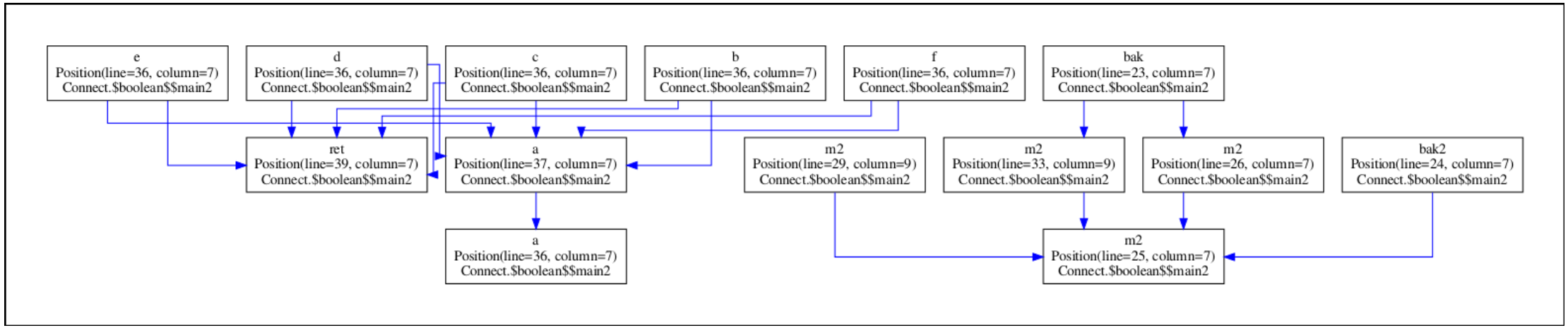
Binding of variables

```
13 package test;
14
15 import android.content.Context;
16 import android.os.AsyncTask;
17
18 public class Connect
19 extends AsyncTask<String, Void, String> {
20
21     public void main2(boolean m)
22     {
23       String bak = "hmmm";
24       String bak2 = "hmmm";
25       String m2 = bak2;
26       m2 = bak;
27       if (m)
28       {
29         m2 = getSystemService("notification");
30       }
31       else
32       {
33         m2 = "dd" + bak;
34       }
35
36       int a, b, c, d, e, f;
37       a = b + c + d + e + f;
38 ......
39       int ret = b + c + d + e + f;
40     }
41 }
42
```

Digital Security

4.2 Taint analysis

Binding of variables



Digital Security

4.2 Taint analysis

This plugin can be used by another plugin

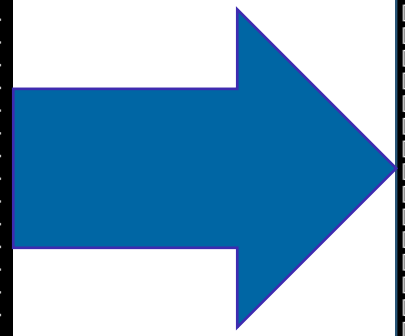
```
 9  class ListFuncs:
10      """
11      Class List functions
12
13      To use:
14        --list_funcs_call '<class>' '<function>'
15        with regex for instance to list all functions:
16        --list_funcs_call '.*' '.*'
17      """
18      def __init__(self, package, tmp_dir, args):
19          load_module("name_file", "name_file_node")
20          load_module("Typer", "typer")
21          load_module("ListFuncsCall", "ListFuncsCalled_node")
22
23          if args.taint:
24              ListFuncsCalled_node.with_taint()
25          ListFuncsCalled_node.Class.set_class(args.list_funcs_call[0])
26          ListFuncsCalled_node.Func.set_func(args.list_funcs_call[1])
27
28          Output.add_printer_callback("tree", "list_funcs_called", "func", mprint)
29
```

Digital Security

4.2 Taint analysis

This plugin can be used by another plugin

```
[ list_funcs_called ] [ func ] toString .....
[ list_funcs_called ] [ func ] toString .....
[ list_funcs_called ] [ func ] Boolean.toString .....
[ list_funcs_called ] [ func ] spannableStringBuilder.toString .....
[ list_funcs_called ] [ func ] toString .....
[ list_funcs_called ] [ func ] xVar.toString .....
[ list_funcs_called ] [ func ] xVar.toString .....
[ list_funcs_called ] [ func ] xVar.toString .....
[ list_funcs_called ] [ func ] bool.toString .....
[ list_funcs_called ] [ func ] Integer.toString .....
[ list_funcs_called ] [ func ] spannableStringBuilder.toString .....
[ list_funcs_called ] [ func ] stringWriter.toString .....
[ list_funcs_called ] [ func ] sb.toString .....
[ list_funcs_called ] [ func ] toString .....
[ list_funcs_called ] [ func ] toString .....
[ list_funcs_called ] [ func ] toString .....
[ list_funcs_called ] [ func ] toString .....
[ list_funcs_called ] [ func ] byteArrayOutputStream.toString .....
[ list_funcs_called ] [ func ] bVar.toString .....
[ list_funcs_called ] [ func ] bVar.toString .....
[ list_funcs_called ] [ func ] toString .....
[ list_funcs_called ] [ func ] a2.toString .....
[ list_funcs_called ] [ func ] eVar.toString .....
[ list_funcs_called ] [ func ] spannableStringBuilder.toString .....
[ list_funcs_called ] [ func ] toString .....
[ list_funcs_called ] [ func ] toString .....
[ list_funcs_called ] [ func ] toString .....
[ list_funcs_called ] [ func ] toString .....
[ list_funcs_called ] [ func ] Objects.toString .....
[ list_funcs_called ] [ func ] Objects.toString .....
[ list_funcs_called ] [ func ] z.PRODUCT_NAME.toString .....
[ list_funcs_called ] [ func ] Objects.toString .....
[ list_funcs_called ] [ func ] Objects.toString .....
[ list_funcs_called ] [ func ] sb.toString .....
[ list_funcs_called ] [ func ] toString .....
[ list_funcs_called ] [ func ] macAddress.toString .....
[ list_funcs_called ] [ func ] toString .....
[ list_funcs_called ] [ func ] toString .....
[ list_funcs_called ] [ func ] sb.toString .....
[ list_funcs_called ] [ func ] bVar.toString .....
[ list_funcs_called ] [ func ] bVar.toString .....
[ list_funcs_called ] [ func ] bVar.toString .....
[ list_funcs_called ] [ func ] spannableString.toString .....
[ list_funcs_called ] [ func ] toString .....
```



```
[ list_funcs_called ] [ func ] Boolean.toString .....
[ list_funcs_called ] [ func ] b.toString .....
[ list_funcs_called ] [ func ] x.toString .....
[ list_funcs_called ] [ func ] x.toString .....
[ list_funcs_called ] [ func ] x.toString .....
[ list_funcs_called ] [ func ] Boolean.toString .....
[ list_funcs_called ] [ func ] StringWriter.toString .....
[ list_funcs_called ] [ func ] StringBuilder.toString .....
[ list_funcs_called ] [ func ] ByteArrayOutputStream.toString .....
[ list_funcs_called ] [ func ] Integer.toString .....
[ list_funcs_called ] [ func ] SpannableStringBuilder.toString .....
[ list_funcs_called ] [ func ] SpannableStringBuilder.toString .....
[ list_funcs_called ] [ func ] toString .....
[ list_funcs_called ] [ func ] toString .....
[ list_funcs_called ] [ func ] DebugMenuAdapter.toString .....
[ list_funcs_called ] [ func ] EditText.toString .....
[ list_funcs_called ] [ func ] EditText.toString .....
[ list_funcs_called ] [ func ] EditText.toString .....
[ list_funcs_called ] [ func ] SpannableString.toString .....
[ list_funcs_called ] [ func ] d.toString .....
[ list_funcs_called ] [ func ] EditText.toString .....
[ list_funcs_called ] [ func ] MacAddress.toString .....
[ list_funcs_called ] [ func ] e.toString .....
[ list_funcs_called ] [ func ] io.intrepid.bose_bmap.model.a.toString
[ list_funcs_called ] [ func ] a1.toString .....
[ list_funcs_called ] [ func ] SpannableStringBuilder.toString .....
[ list_funcs_called ] [ func ] n.toString .....
[ list_funcs_called ] [ func ] n.toString .....
[ list_funcs_called ] [ func ] n.toString .....
[ list_funcs_called ] [ func ] StringBuilder.toString .....
[ list_funcs_called ] [ func ] MacAddress.toString .....
[ list_funcs_called ] [ func ] io.toString .....
[ list_funcs_called ] [ func ] d.toString .....
[ list_funcs_called ] [ func ] HeartRateView.toString .....
[ list_funcs_called ] [ func ] HeartRateView.toString .....
[ list_funcs_called ] [ func ] HeartRateView.toString .....
[ list_funcs_called ] [ func ] Objects.toString .....
[ list_funcs_called ] [ func ] Objects.toString .....
[ list_funcs_called ] [ func ] z.PRODUCT_NAME.toString .....
[ list_funcs_called ] [ func ] Objects.toString .....
[ list_funcs_called ] [ func ] Objects.toString .....
[ list_funcs_called ] [ func ] toString .....
[ list_funcs_called ] [ func ] toString .....
[ list_funcs_called ] [ func ] StringBuilder.toString .....
```

Digital Security

4.2 Generate APK

- ASTHOOK can generate some APK with a Jinja2 template.
- A folder is generated with all the source code needed.

Digital Security

4.2 Generate APK

Make your template

File: **asthook/static/module/VulnIntent/poc/java/exploit/intent/exploit.java**

```
1  package exploit.intent;
2
3  import android.app.Activity;
4
5  import android.os.Bundle;
6  import android.util.Log;
7  import android.content.Intent;
8  import android.content.ComponentName;
9  import android.net.Uri;
10
11 public class exploit extends Activity {
12
13     @Override
14     protected void onCreate(Bundle savedInstanceState) {
15         super.onCreate(savedInstanceState);
16         Intent intent = new Intent();
17         intent.setClassName("{{ app }}", "{{ activity }}");
18         {% for item in parameters %}
19         intent.putExtra({{ item.name }}, {{ item.value }});
20         {% endfor %}
21         {% if data|length %}
22         intent.setData(Uri.parse("{{ data }}"));
23         {% endif %}
24         {% if datas|length %}
25         intent.setClipData(ClipData.newRawUri(null, "{{ datas }}"));
26         {% endif %}
27         startActivity(intent);
28         finish();
29     }
30 }
```

Digital Security

4.2 Generate APK

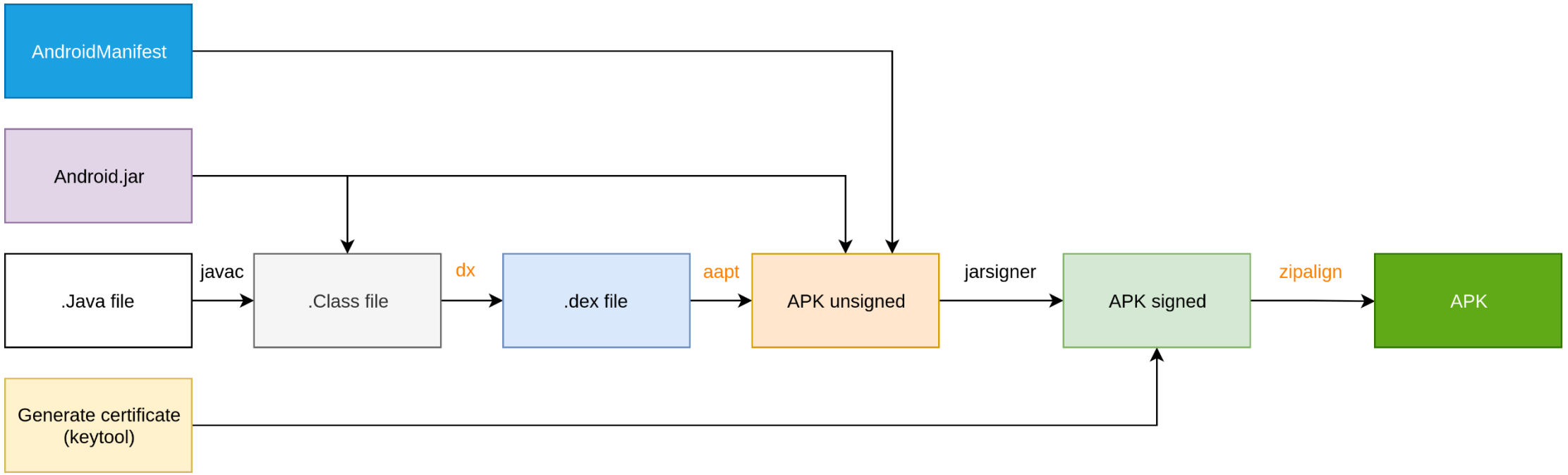
Call your template with variables and make the APK

```
manifest = JavaFile("/AndroidManifest.xml",
        path + "AndroidManifest.xml",
        {'app' : app,
         'activity' : activity})
exploit = JavaFile("/exploit/intent/exploit.java",
        path + "/java/exploit/intent/exploit.java",
        {'app' : app,
         'activity' : activity,
         'parameters' : parameters,
         'data': Data,
         'datas': Datas})
GenerateAPK("vulnIntent_%s_%s" % (activity, k),
        manifest,
        [exploit],
        self.args, self.get_tmp())
```

Digital Security

4.2 How an APK is made ?

Magic Makefile



Digital Security

4.3 Dynamic analysis

Instrumentation

Digital Security

4.3 Intercept https traffic with a proxy

Set up the proxy server to intercept https was fastidious...

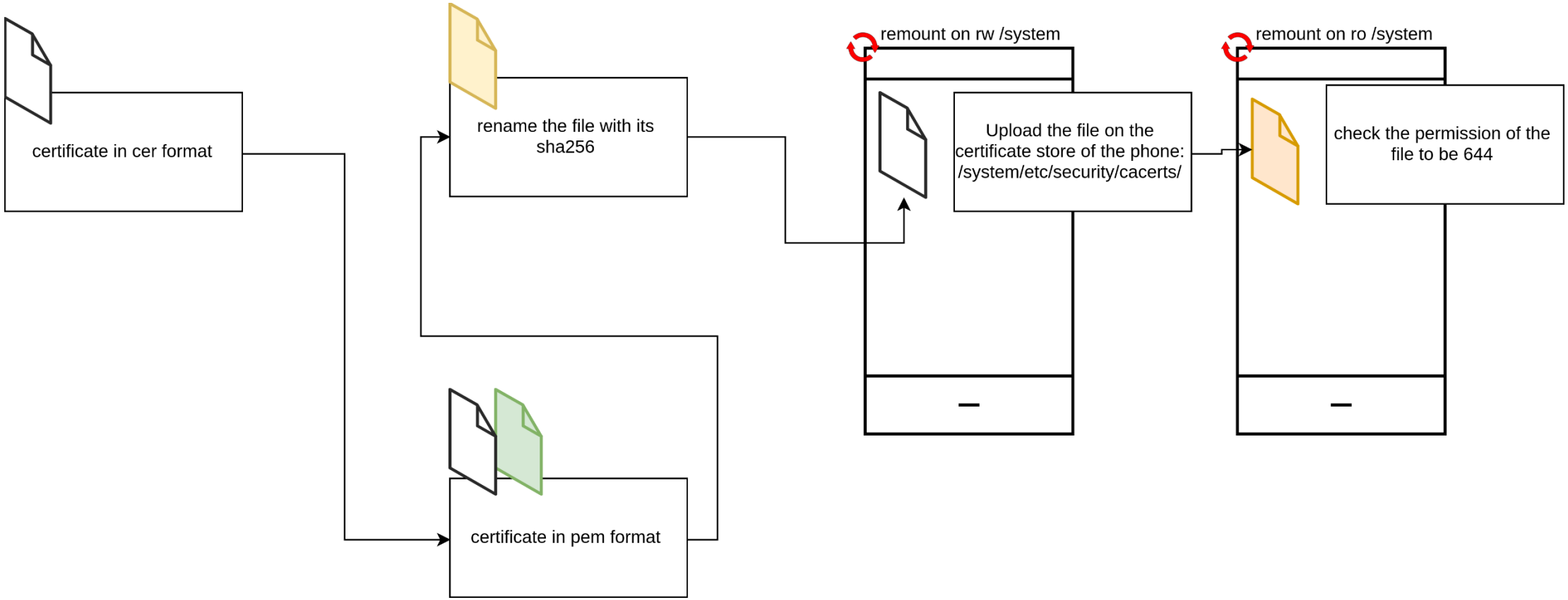
...but not anymore with Asthook.

1. Download your proxy's certificate
2. Passes it as an argument of Asthook
3. Wait for your first interception

Digital Security

4.3 Intercept https traffic with a proxy

How it works ?

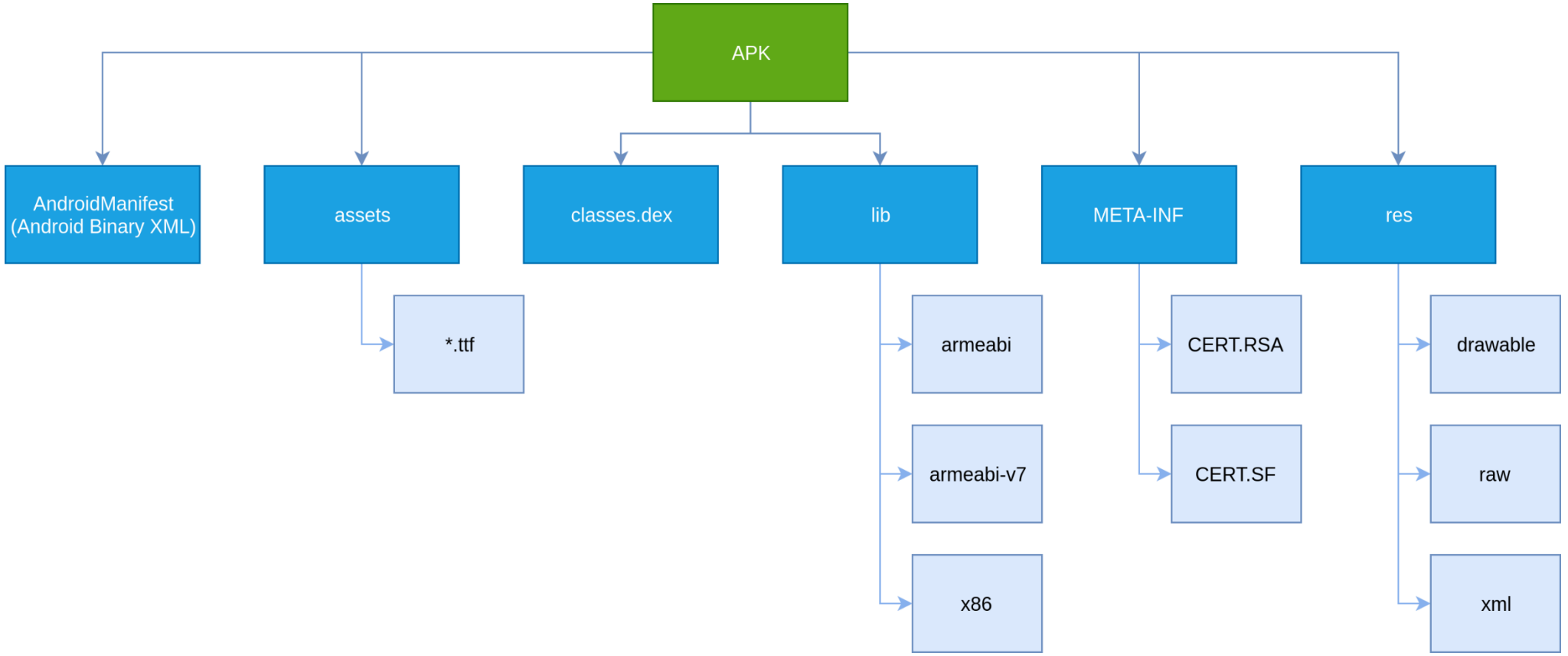


And what if the phone is not rooted ?

Digital Security

4.3 Not Rooted Phone is not a problem

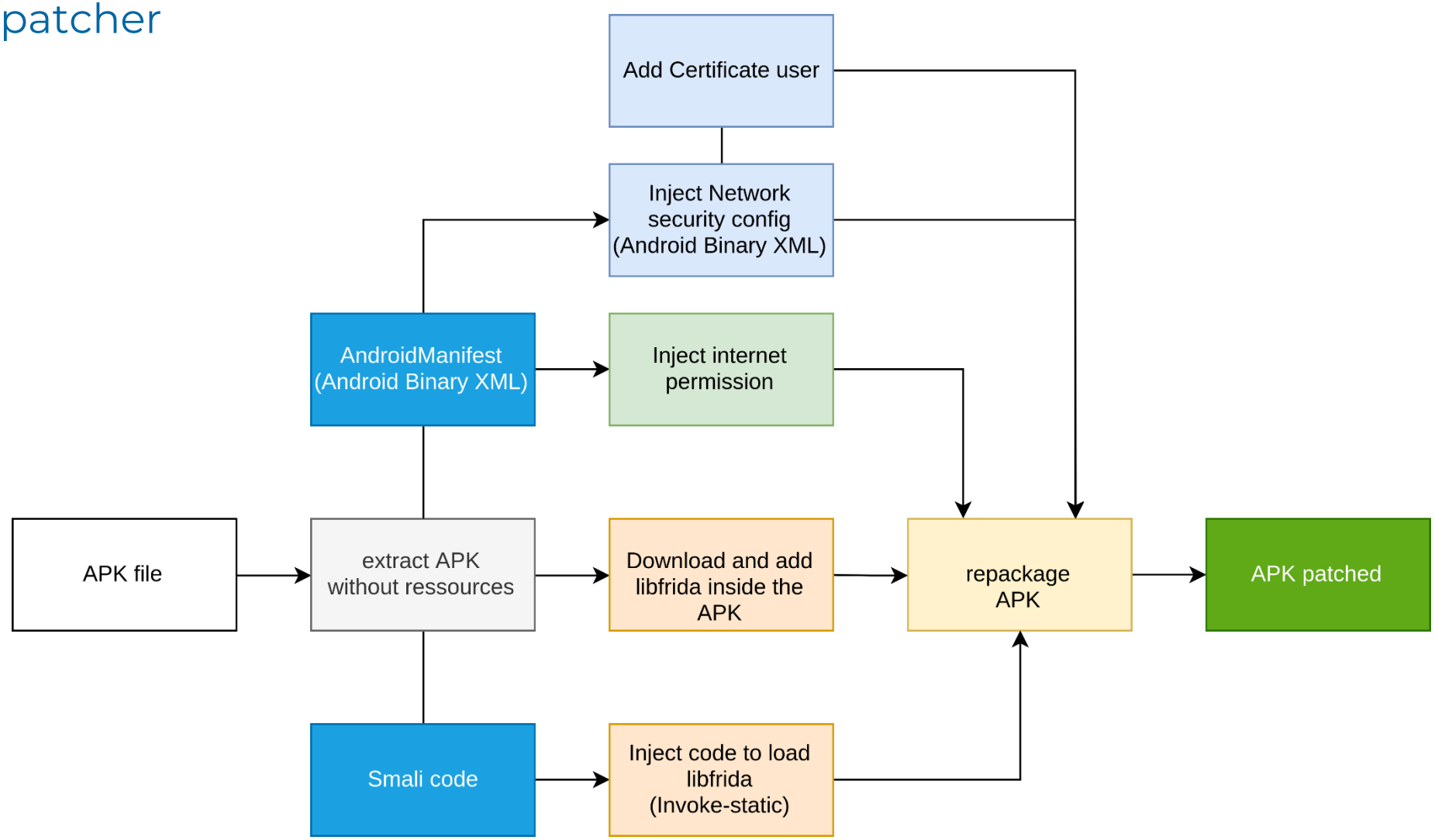
Apkpatcher



Digital Security

4.3 Not Rooted Phone is not a problem

Apkpatcher



Plugin interface for dynamic analysis

Digital Security

4.3 Plugin interface for dynamic analysis

- ADB interface (shell, push, pull, install, spawn etc.)
- Frida (load, unload) + interaction with python codes

6. What is next ?

Q&A ?

Principal repo: https://gitlab.com/MadSquirrels/mobile/asthook
Secondary repo: https://github.com/Mrbenoit624/ASThook

Digital Security

info@digital.security

+33(0)1 70 83 85 85

@iotcert

Digital Security by ATOS

www.digital.security