

Static & Dynamic Analysis

ACN / Mobile Security 2020

Johannes Feichtner
johannes.feichtner@iaik.tugraz.at

Outline

- Why app analysis?
- Approaches / Techniques
 - Reverse Engineering
 - Static & Dynamic Program Inspection
- Tools



Introduction

Problem

- Software that is cheap, correct, secure, efficient, ... is a myth!
- Users have to **trust** (unknown) developers
- Source code usually not published publicly
 - How many % of app users could verify it anyway?

What now?

- Minimize number of security threats
 - Developers will always make mistakes
 - Need for automated security checks
- Software reverse-engineering



CHALLENGE 1: Broader mobile attack surface

THE DEVICE



BROWSER

- Phishing
- Framing
- Clickjacking
- Man-in-the-Middle
- Buffer overflow
- Data caching



PHONE/SMS

- Baseband attacks
- SMS phishing



SYSTEM

- No/Weak passcode
- Android rooting/iOS jailbreak
- OS data caching
- Passwords & data accessible
- Carrier-loaded software
- No/Weak encryption
- User-initiated code
- Confused deputy attack
- TEE/Secure Enclave Processor
- Side channel leak
- Multimedia/file format parsers
- Kernel driver vulnerabilities
- Resource DoS
- GPS spoofing
- Device lockout



APPS

- Sensitive data storage
- No/Weak encryption
- Improper SSL validation
- Configuration manipulation
- Dynamic runtime injection
- Unintended permissions
- Escalated privileges
- UI overlay/pin stealing
- Third-party code
- Intent hijacking
- Zip directory traversal
- Clipboard data
- URL schemes
- GPS spoofing
- Weak/No Local authentication
- Integrity/tampering/repacking
- Side channel attacks
- App signing key unprotected
- App transport security
- XML serialization
- JSON-RPC
- SQLite database



MALWARE

THE NETWORK

- Wi-Fi (no/weak encryption)
- Rogue access point
- Packet sniffing
- Man-in-the-middle
- Session hijacking
- DNS poisoning
- SSL Strip
- Fake SSL certificate

- Baseband
- Wifi (chip/firmware attack)
- BGP hijacking
- IMSI-catcher
- LTE
- HTTP Proxies
- VPNs



CLOUD / DATA CENTER

WEB SERVER

- Platform vulnerabilities
- Server misconfiguration
- Cross-site scripting
- Cross-site request forgery
- Weak input validation
- Cross origin resource sharing
- Brute force attacks
- Side channel attacks
- Hypervisor attack
- VPN

DATABASE

- SQL injection
- Privilege escalation
- Data dumping
- OS command execution



OWASP

Open Web Application
Security Project

WWW.OWASP.ORG

Introduction

Technical objectives

- Fight complexity
 - More code = more bugs = more attack vectors
- Secure platform
 - Smaller attack surface for apps
 - Tighter boundaries for data misuse
- Find out what apps are doing
 - Reverse engineering
 - How do know what they are doing?
 - What is malware?

Reverse Engineering

discovering the technological principles of a device, object or system through analysis of its structure, function and operation

Source: <https://goo.gl/UZqNm>

Why?

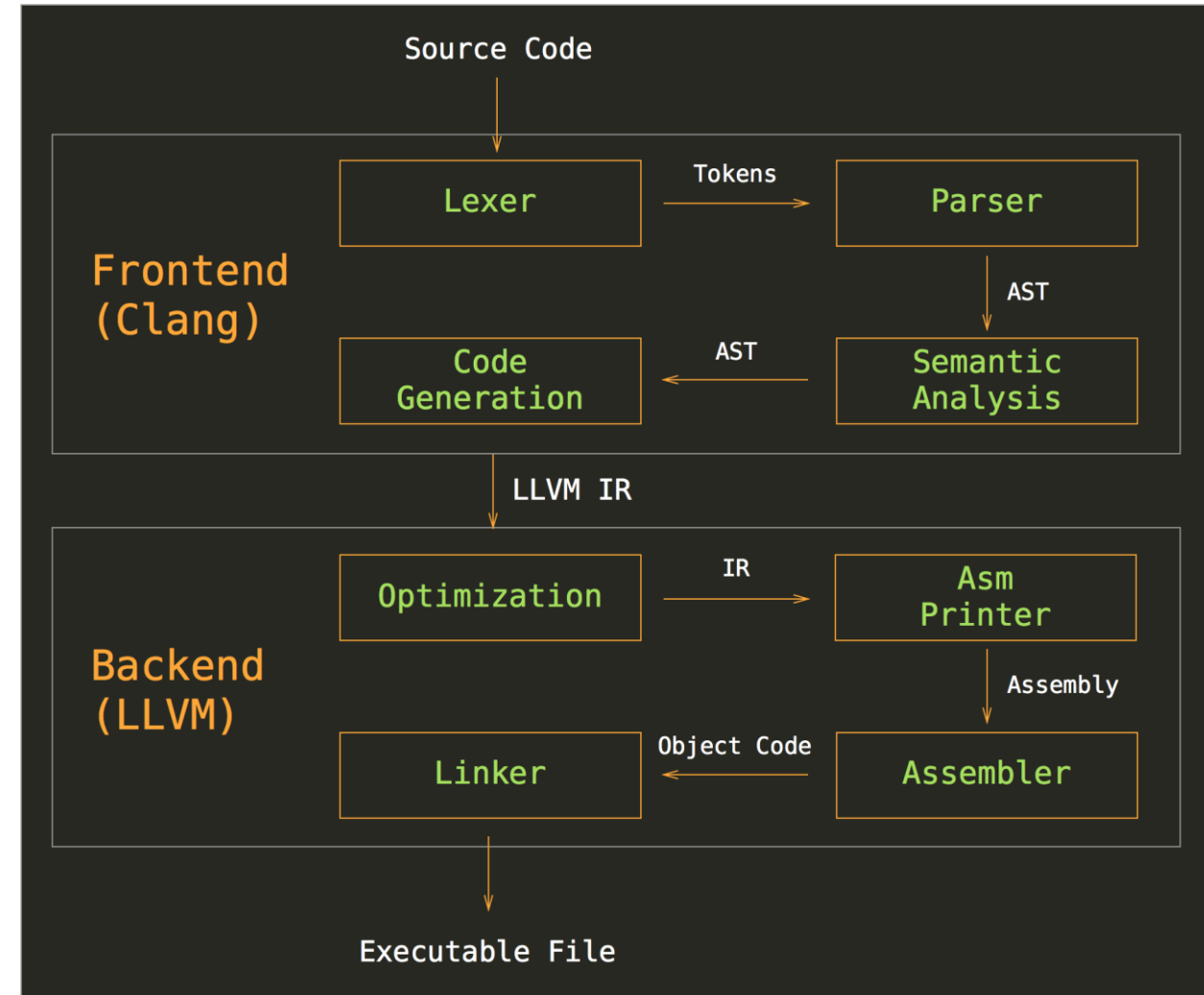
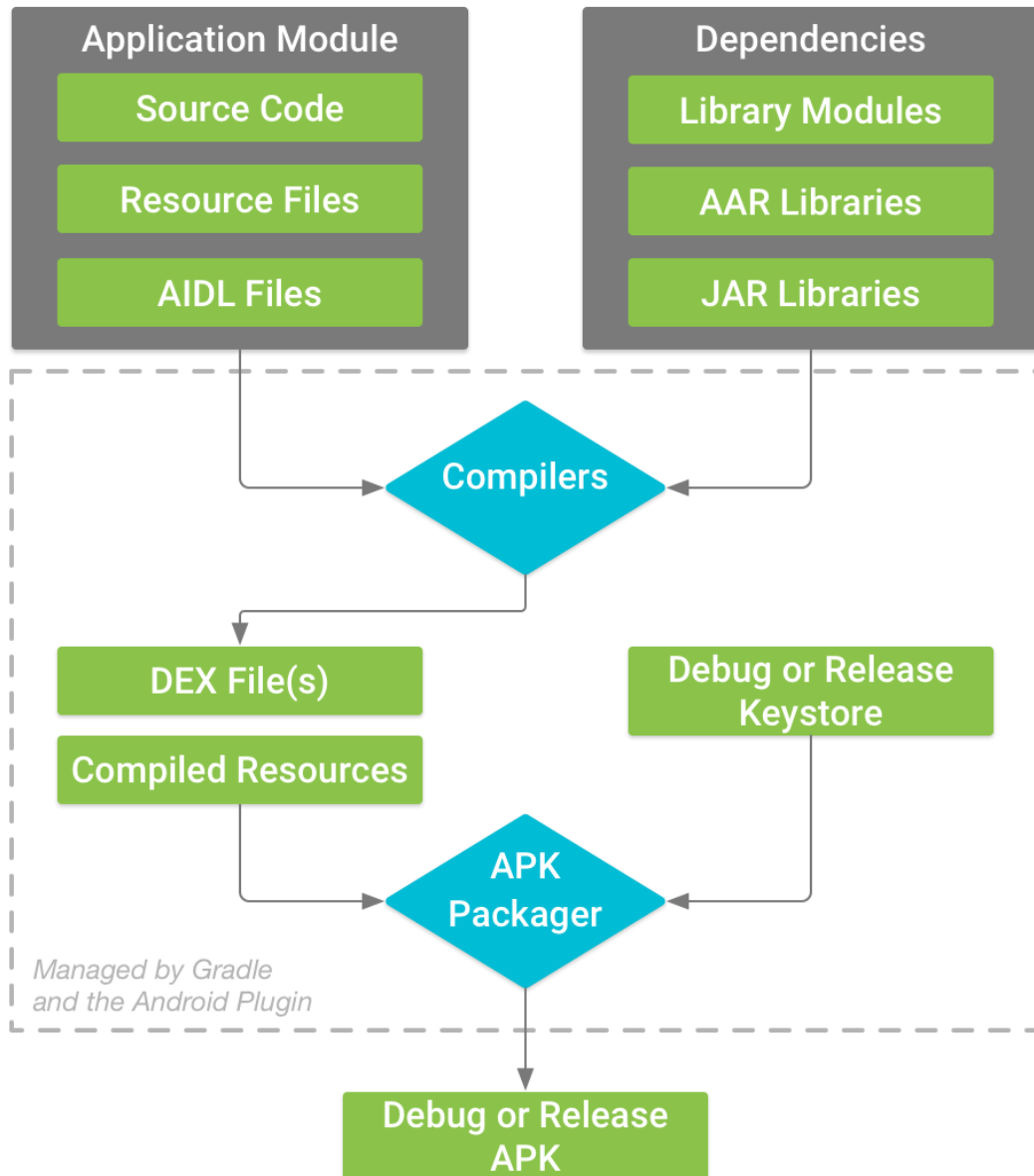
- Curiosity :-)
- Protocol interoperability
 - Windows file share support in Linux (Samba)
- API compatibility
 - Windows emulation on Linux (Wine)
- Unlocking hardware
 - Jailbreaking iPhone, PS4

State of Mobile Application Security

Different Types

- Static Analysis
 - Analyze code without running
 - Inspecting disassembled or decompiled code
- Dynamic Analysis
 - Analyze app behaviour while running
 - Network traffic, file system access, user input, sensor usage, ...
 - Mostly done in emulators
- Hybrid Analysis
 - Combines both types and often multiple tools

Build Processes



Static Analysis

Control Flow Graph

What?

- All possible execution paths in a program
- Directed graph consisting of
 - Nodes = Basic Blocks
 - Edges = Possible flow between nodes

Steps to obtain CFG

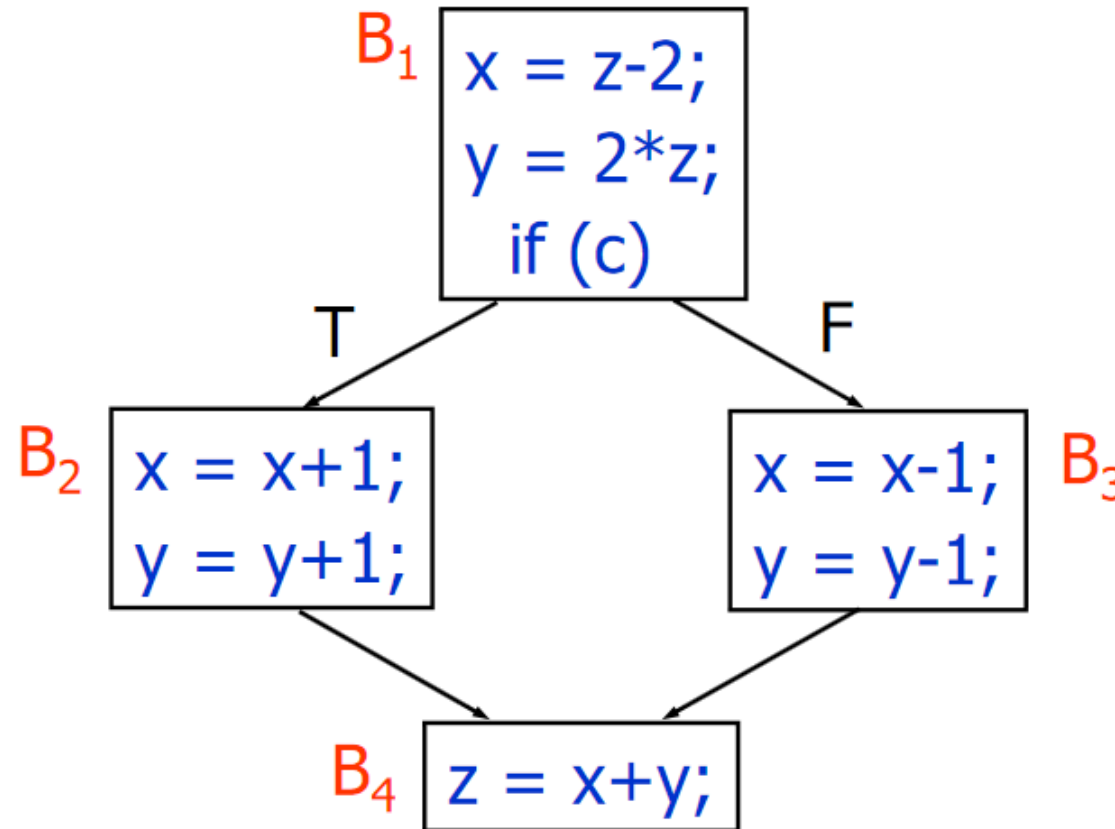
1. Identify all basic blocks
 - Instructions that cannot halt oder branch out
2. Add all edges
 - Hard with indirect calls / self-modifying code

Control Flow Graph

Program

```
x = z-2 ;  
y = 2*z;  
if (c) {  
    x = x+1;  
    y = y+1;  
}  
else {  
    x = x-1;  
    y = y-1;  
}  
z = x+y;
```

Control Flow Graph



Data Flow Graph

What?

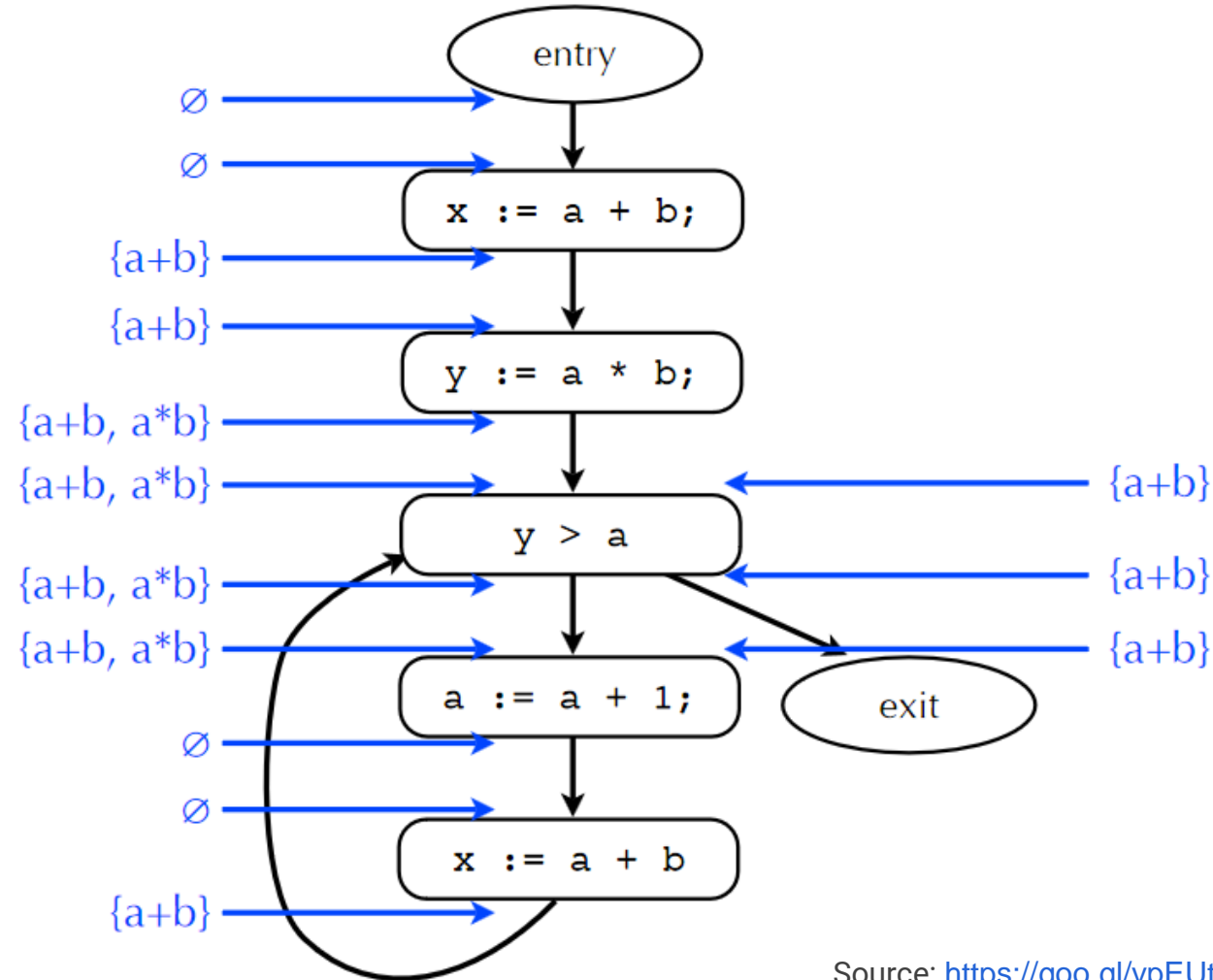
- Get all variable changes at some execution point
- Shows show data changes through a Basic Block
 - What input/output is sent to/from a function?

Two types

- Forward Analysis
 - „Find all statements that are **influenced** by some starting point (slicing criterion)“
- Backward Analysis
 - „Find all statements that are **influencing** some target (slicing criterion)“

Forward Analysis

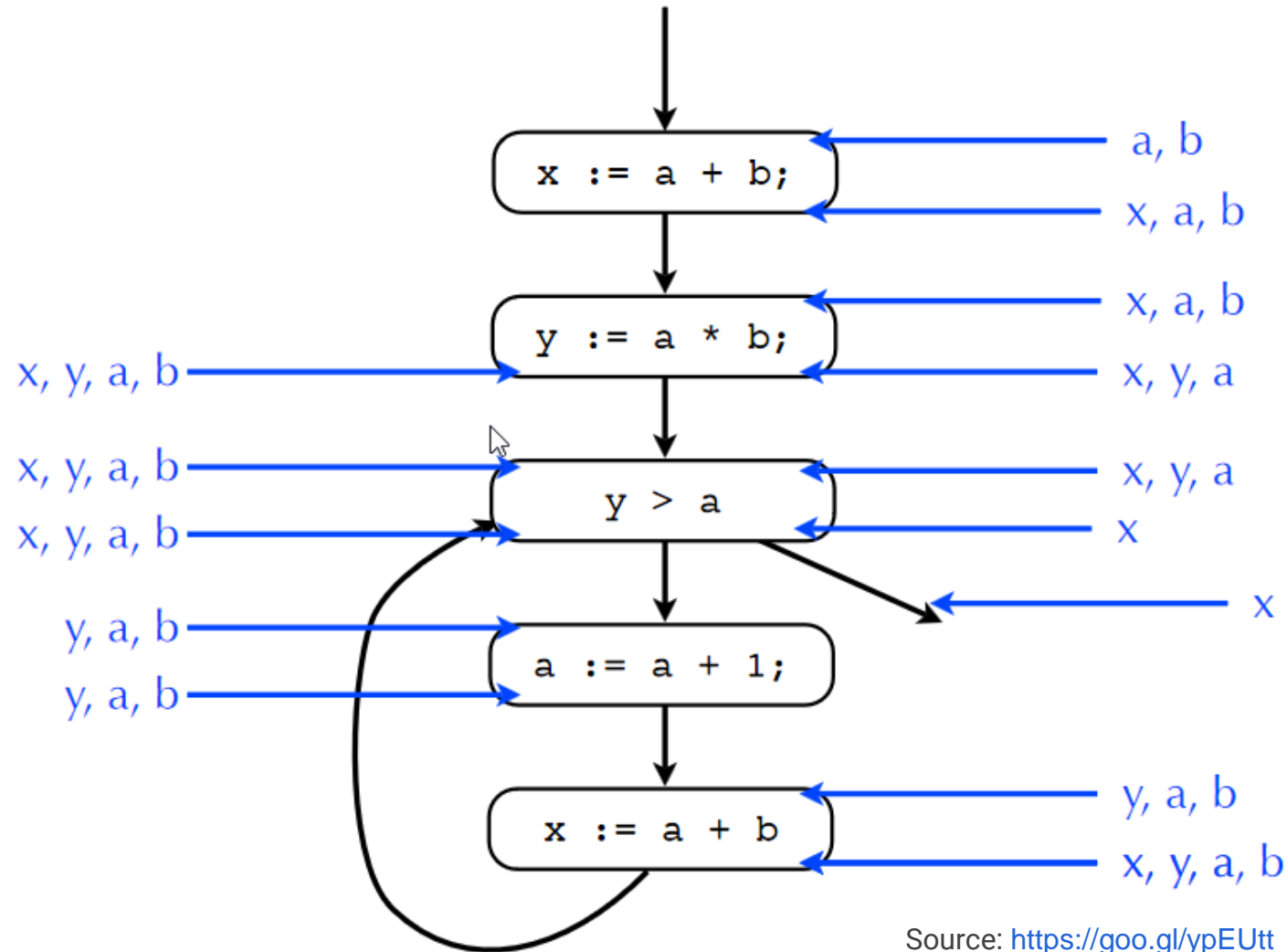
Computing available expressions



Source: <https://goo.gl/ypEUtt>

Backward Analysis

Computing live variables



Source: <https://goo.gl/ypEUtt>

Source & Sink Analysis

What?

- **Source** = User's location, address book, camera
 - **Sink** = Internet, SMS, Bluetooth, ...
- Check if there is potential data flow between source & sink

On Android / iOS

- Sensitive sources reachable via API methods, e.g. cell location
- Hundreds of possible sources and sinks

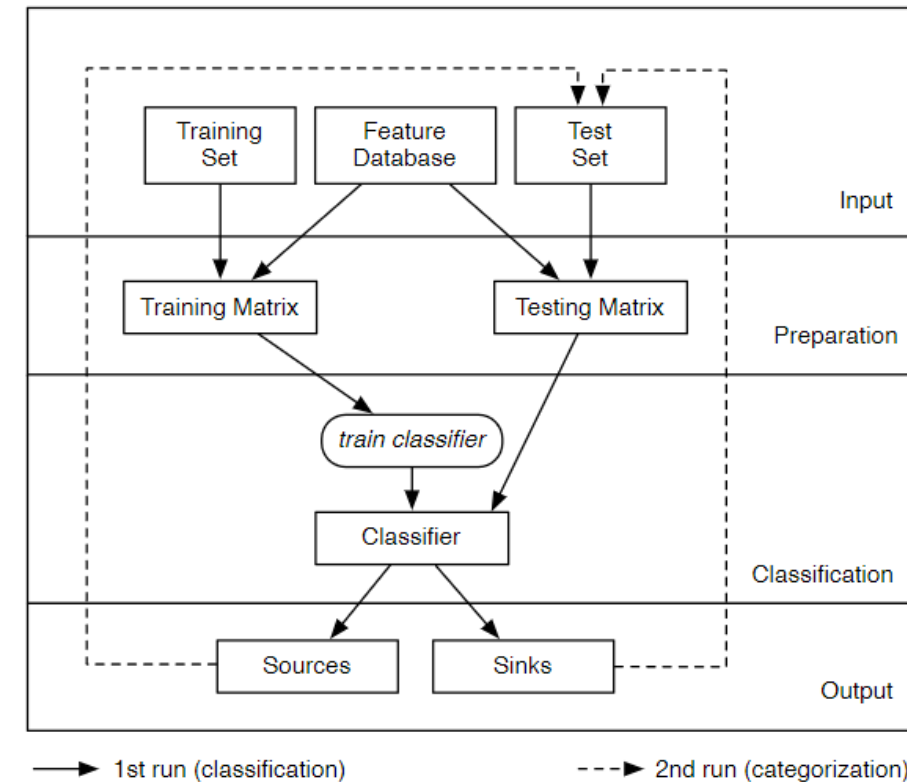
```
1 void onCreate() {
2   TelephonyManager tm; GsmCellLocation loc;
3   // Get the location
4   tm = (TelephonyManager) getContext().
5     getSystemService
6     (Context.TELEPHONY_SERVICE);
7   loc = (GsmCellLocation)
8     tm.getCellLocation();
9
10  //source: cell-ID
11  int cellID = loc.getCid();
12  //source: location area code
13  int lac = location.getLac();
14  boolean berlin = (lac == 20228 && cellID
15    == 62253);
16
17  String taint = "Berlin: " + berlin + " ("
18    + cellID + " | " + lac + ")";
19  String f = this.getFilesDir() +
20    "/mytaintedFile.txt";
21  //sink
22  FileUtils.stringToFile(f, taint);
23  //make file readable to everyone
24  Runtime.getRuntime().exec("chmod 666 "+f);
25 }
```

Source: <https://goo.gl/6o2VU2>

Android - SUSI



- Supervised Machine Learning
 - Train classifier with small set of manually defined APIs as sources
 - Apply on whole Android source code code to find other sources and sinks
- Outputs lists of possible sources and sinks
 - Does not find leaks by itself
 - Can be used as basis for taint tracking



Source: <https://goo.gl/6o2VU2>

Android – Soot

What?

- Started as Java optimization framework
 - Now used to analyse Java / Android, optimize, visualize

Features

- Call-graph reconstruction
 - Calling relationships between subroutines
- Points-to analysis
 - Which pointers or heap references can point to which variables / storage locations
- Def-use chains
 - Forward Analysis
- Data flow analysis

Dynamic Analysis

Workflow

What?

- Analysis of properties of running program
- Only parts of programs that are actually executed
 - No code snippets
 - Usually run in sandbox / emulator
- Black-box testing

Purpose

- Run-time error detection
- Test program behaviour with user interactions
- Check for malicious / strange actions

Workflow

On Android / iOS

Network traffic, IPC, Permission usage, Accessed resources, Sensor data

Definable

- Environment
 - Virtual Machine / Emulator: Easier to monitor and reset
 - Physical device: Real sensor data, WiFi networks, etc.
- Logging
 - Create protocol while running
- Interaction
 - Simulate user input
- Execution time

Android - Droidbox

- Dynamic Taint Analysis and Method hooking
- Needs modified Android version
 - Patches Dalvik and core system

Analyzes

- Incoming / outgoing traffic
- File read, write operations
- Listing broadcast receivers
- Sent SMS and phone calls
- Performed cryptographic operations

```
23 [Read operations]
24 -----
25 [22.9400451183] Path: /data/data/droidbox.tests/files/myfilename.txt^A
26 Data: Write a line
27 [24.2107310295] Path: /data/data/droidbox.tests/files/myfilename.txt
28 Data:
29 [25.997330904] Path: /data/data/droidbox.tests/files/output.txt
30 Data: null
31 [26.781430006] Path: /data/data/droidbox.tests/files/output.txt
32 Data:
33 [Write operations]
34 -----
35 [21.3330090046] Path: /data/data/droidbox.tests/files/myfilename.txt^A
36 Data: Write a line
37 [21.3614990711] Path: /data/data/droidbox.tests/files/output.txt
38 Data: null
39 [Crypto API activities]
40 -----
41 [26.8029410839] Key:{0, 42, 2, 54, 4, 45, 6, 7, 65, 9, 54, 11, 12, 13, 60, 15} Algorithm: AES
42 [26.811686039] Operation:{encryption} Algorithm: AES
43 Data:{357242043237517}
44 [26.818600893] Key:{0, 42, 2, 54, 4, 45, 6, 7, 65, 9, 54, 11, 12, 13, 60, 15} Algorithm: AES
45 [26.8250999451] Operation:{decryption} Algorithm: AES
46 Data:{357242043237517}
47 [26.8305909634] Key:{0, 42, 2, 54, 4, 45, 6, 8} Algorithm: DES
48 [26.8399989605] Operation:{encryption} Algorithm: DES
49 Data:{357242043237517}
50 [26.8453080654] Key:{0, 42, 2, 54, 4, 45, 6, 8} Algorithm: DES
51 [26.853967905] Operation:{decryption} Algorithm: DES
52 Data:{357242043237517}
```

Android / iOS - Frida

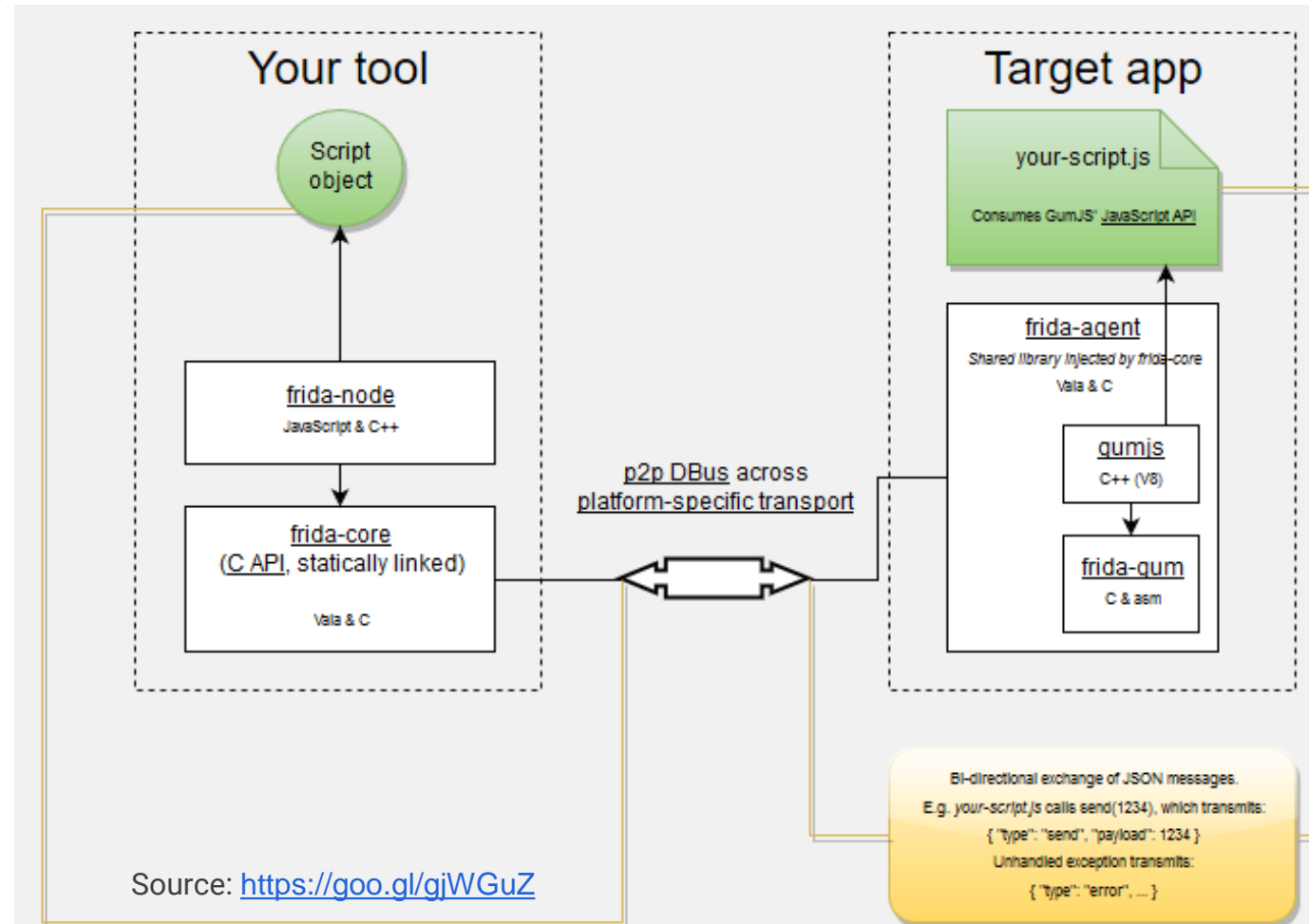
FRIDA

Principle

1. Inject custom logic into process
2. Intercept function calls
3. Stalk process
 - Code tracing
 - Avoid anti-debugger products

Features

- Attaching to process
- Hooking & calling functions
- Modifying function arguments
- Inspecting & modifying memory



Outlook

- 04.06.2020
 - Mobile Network Security

- 18.06.2020
 - Assignment Presentations