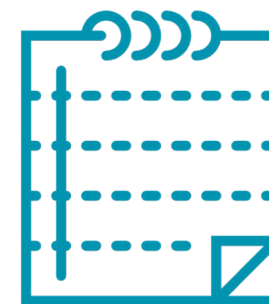


The Mobile Threat Landscape

Luca Capacci
Bologna, 24/05/2018

AGENDA (1)

MOBILE MALWARE

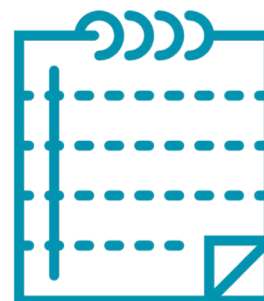


- Mobile Landscape
- Desktop Malware vs Mobile Malware
- Android Security Model
- iOS Security Model
- Malware Types
- Malware Analysis Techniques

AGENDA (2)

MOBILE APP SECURITY

- Web Threats vs Mobile Threats
- Mobile Security Testing
- OWASP Top Ten Mobile



Part 1

An overview of
malware for Android
and iOS

Mobile Landscape

- Location-independent (mobile)
- “Always online” and traceable
- Consumerization – devices are built for personal use
- Focus on functionality and design rather than security
- Raise of sensitive use cases for mobile apps



https://www.owasp.org/images/0/04/Security_Testing_Guidelines_for_mobile_Apps_-_Florian_Stahl%2BJohannes_Stroeher.pdf

Desktop Malware vs Mobile Malware

- Mobile operating systems have stronger security measures than desktop (unless you jailbreak your phone...)

BUT

- There are more mobile devices than PCs!

Desktop Malware vs Mobile Malware

 BUSINESS INSIDER

TECH INSIDER

IT'S OFFICIAL: The Smartphone Market Is Now Bigger Than The PC Market

 Henry Blodget   
Feb. 8, 2011, 9:40 PM  13,356

 FACEBOOK  LINKEDIN  TWITTER  EMAIL  PRINT

Well, that was fast.

Only three years after the iPhone launched, the [smartphone market](#) is already bigger than the PC market.

According to IDC, 101 million smartphones were sold in Q4, versus 92 million PCs.

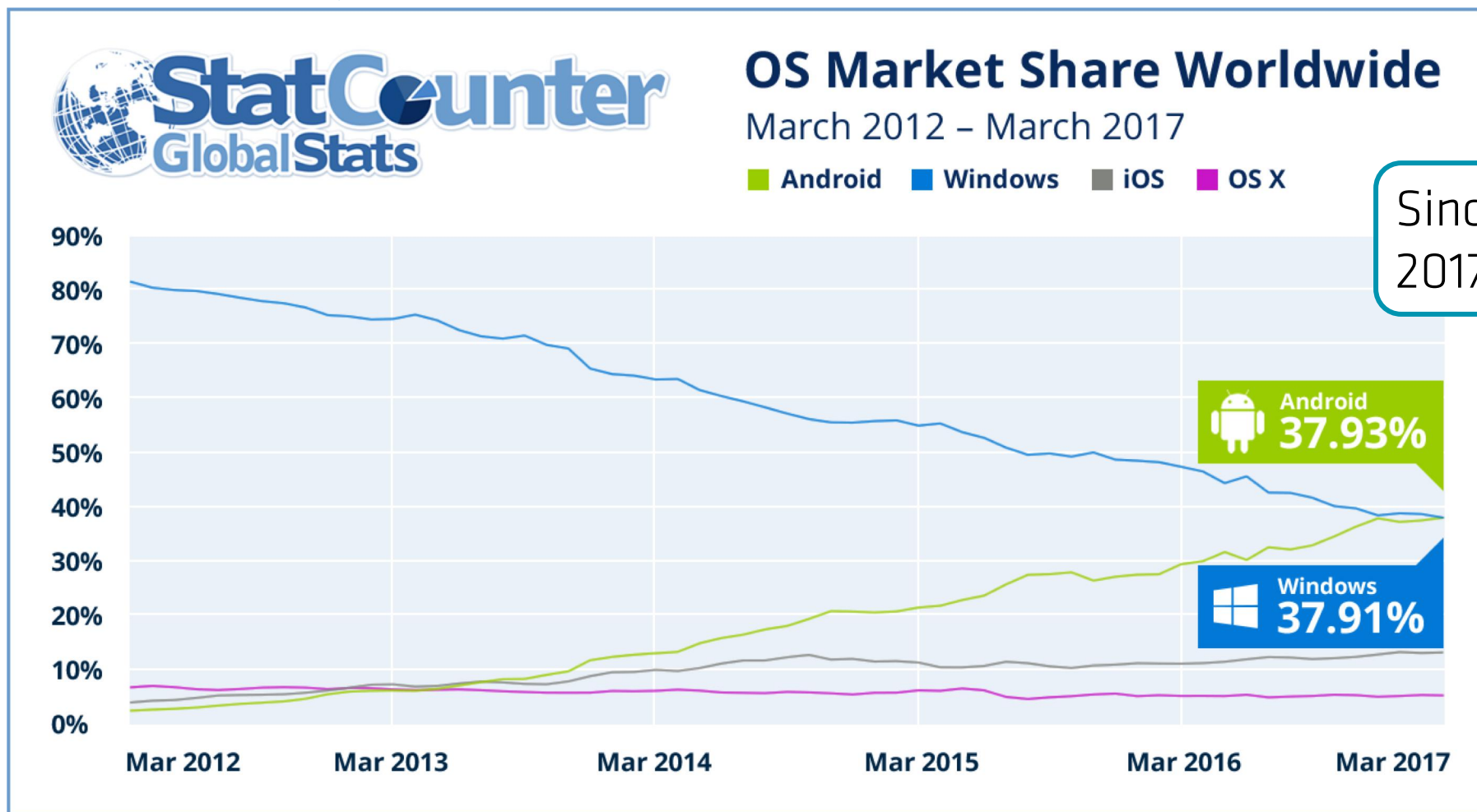
Smartphone shipments, meanwhile, grew 87% year over year, while PCs only grew 3%.



Since
2011

<http://www.businessinsider.com/smartphone-bigger-than-pc-market-2011-2?IR=T>

Desktop Malware vs Mobile Malware



<http://gs.statcounter.com/press/android-overtakes-windows-for-first-time>

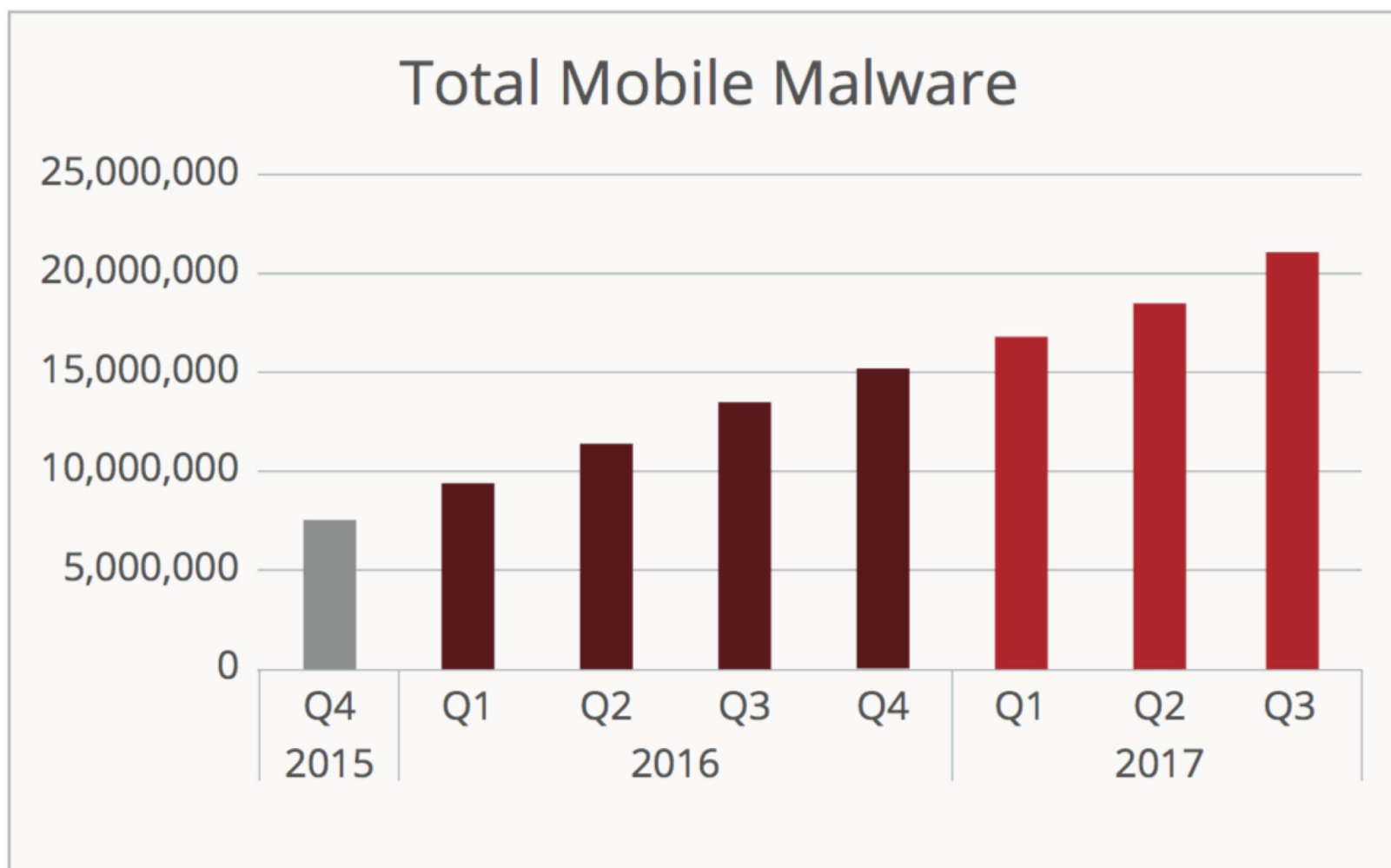
Mobile Malware



2012: first malware found in both Android Google Play and iOS App Store

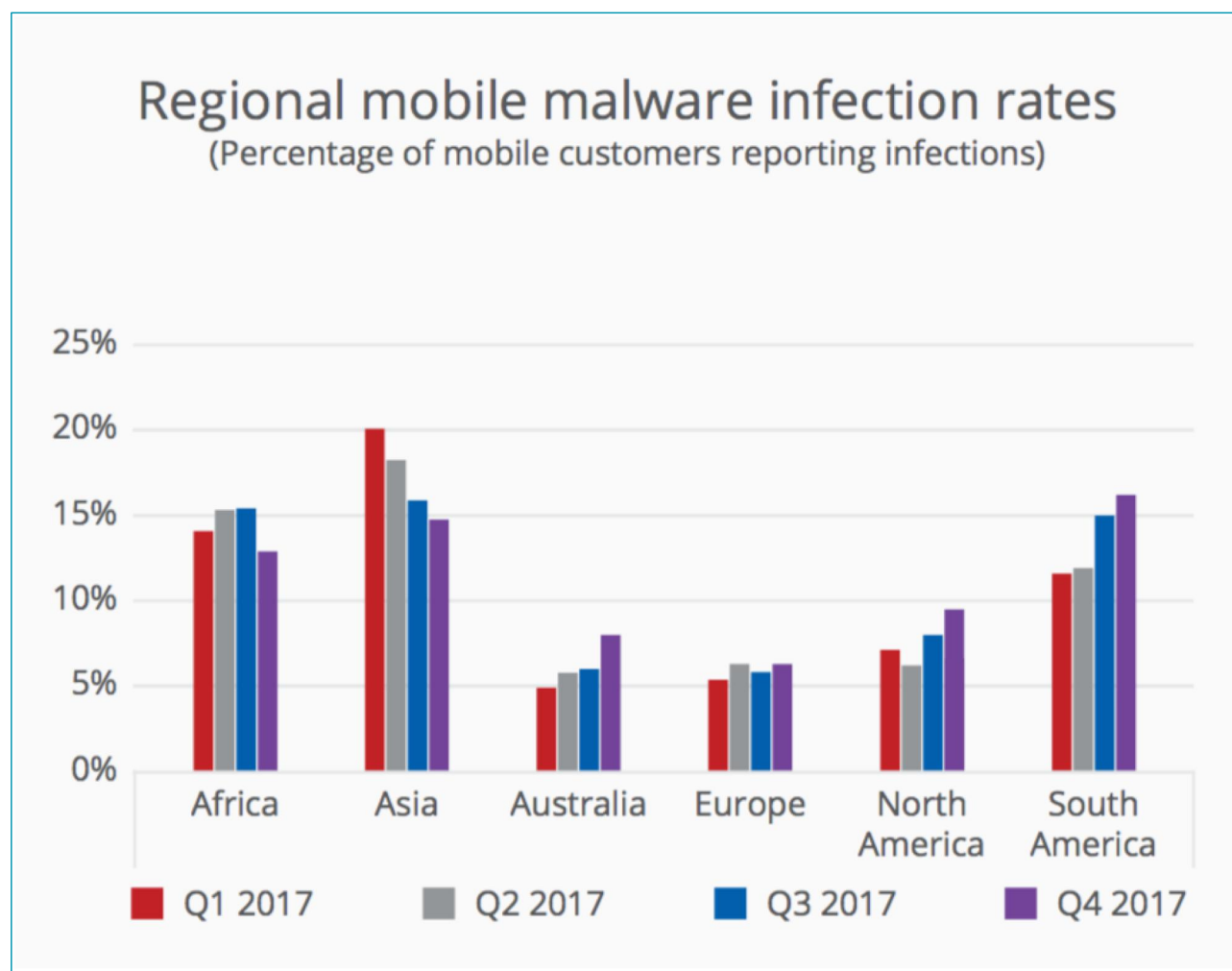
<https://securelist.com/blog/incidents/33544/find-and-call-leak-and-spam-57/>

Total Mobile Malware



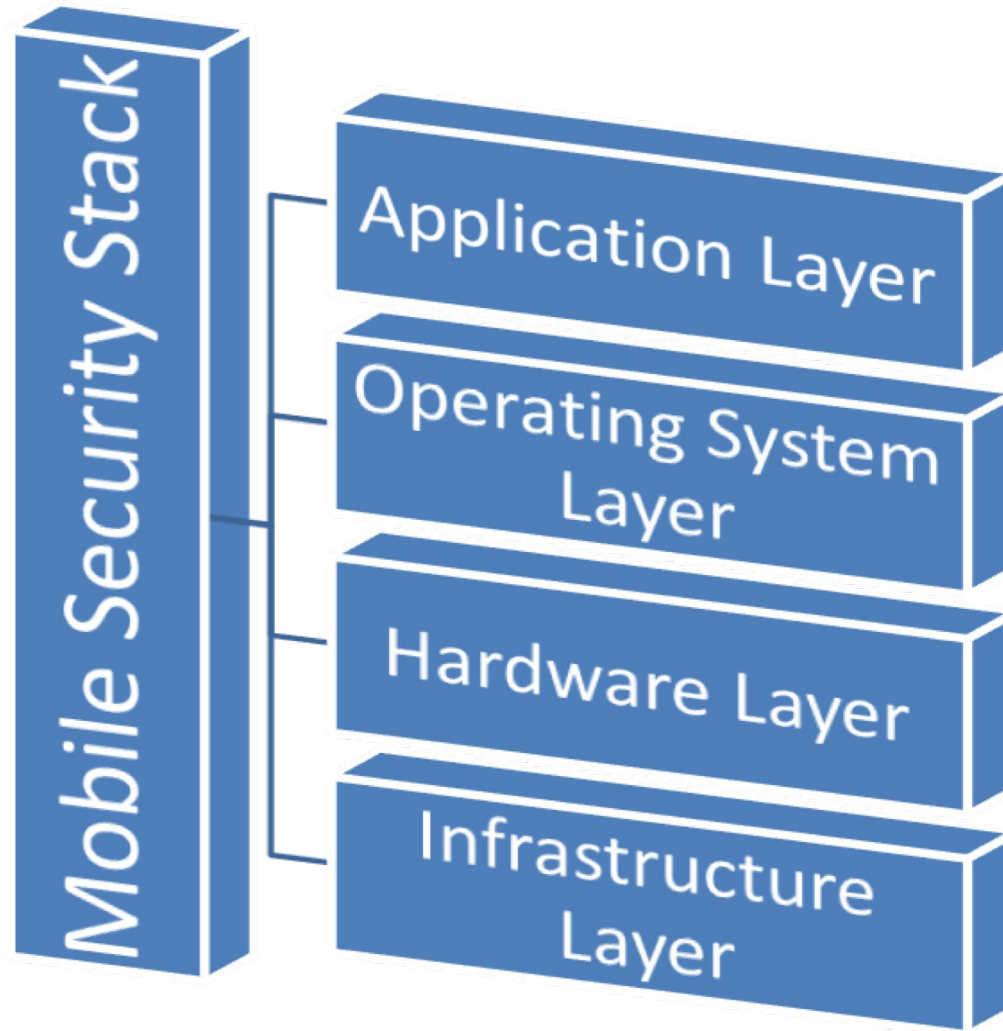
<https://www.mcafee.com/cn/resources/reports/rp-mobile-threat-report-2018.pdf>

Mobile Malware Distribution



<https://www.avency-security.de/wp-content/uploads/2018/03/rp-quarterly-threats-mar-2018-1.pdf>

The Mobile Code Security Stack



Upper layers rely on the security of lower layers

<https://www.veracode.com/security/mobile-code-security>

Infrastructure Layer

- Cellular network: responsibility of cellular voice and data carriers and infrastructure providers
- Security of the protocols in use. Examples:
 - code division multiple access protocol (CDMA)
 - global system for mobile communications (GSM)
 - global positions systems (GPS)
 - short messaging systems (SMS)
 - multimedia messaging systems (MMS)
- Flaws or vulnerabilities discovered at this tier are generally effective across multiple platforms, multiple carriers and multiple handset providers

Infrastructure Layer

- Signalling System No. 7 (SS7): a set of telephony signaling protocols developed in 1975, which is used to set up and tear down most of the world's public switched telephone network (PSTN) telephone calls

After years of warnings, mobile network hackers exploit SS7 flaws to drain bank accounts

O2 confirms online thefts using stolen 2FA SMS codes



3 May 2017 at 20:02, Iain Thomson



Experts have been warning for years about security blunders in the Signaling System 7 protocol – the magic glue used by cellphone networks to communicate with each other.

http://www.theregister.co.uk/2017/05/03/hackers_fire_up_ss7_flaw
https://en.wikipedia.org/wiki/Signalling_System_No._7

Hardware Layer

- End user premise equipment, generally in the form of a smartphone or tablet style mobile device: typically under the direct control of the user
- Managed and used by the operating system
- Security flaws or vulnerabilities discovered at this layer typically affect all end users who use a particular piece of hardware or individual hardware component

Hardware Layer

QuadRouter: New Android Vulnerabilities in Over 900 Million Devices

by Adam Donenfeld, Check Point Mobile Research Team

posted 2016/08/07

2016



Check Point today disclosed details about a set of four vulnerabilities affecting 900 million Android smartphones and tablets that use Qualcomm® chipsets. The Check Point mobile threat research team, which calls the set of vulnerabilities QuadRouter, presented its findings in a session at DEF CON 24 in Las Vegas.

What is QuadRouter?

QuadRouter is a set of four vulnerabilities affecting Android devices built using Qualcomm chipsets. Qualcomm is the world's leading designer of LTE chipsets with a 65% share of the LTE modem baseband market. If any one of the four vulnerabilities is exploited, an attacker can trigger privilege escalations for the purpose of gaining root access to a device.

<http://blog.checkpoint.com/2016/08/07/quadrouter/>

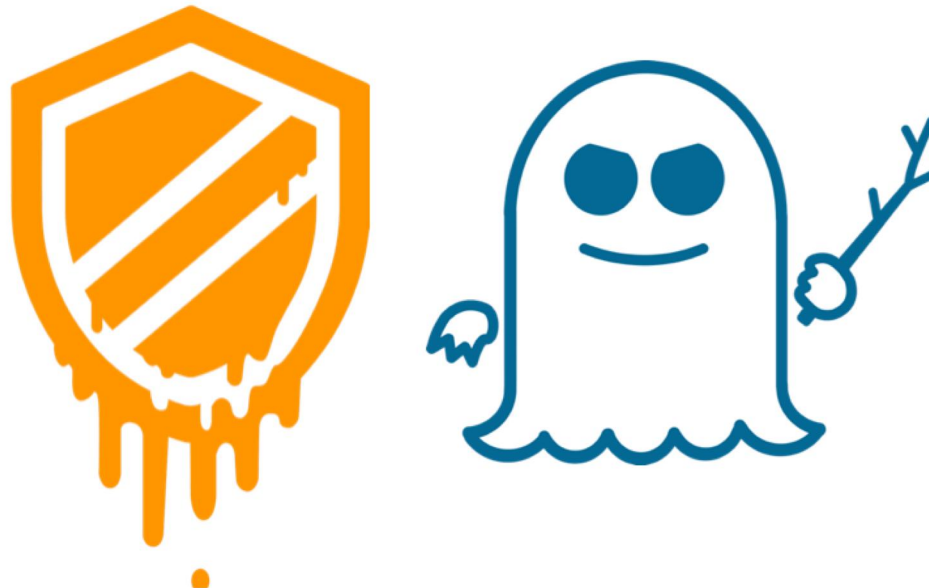
Hardware Layer

Spectre & Meltdown: How 2 of the largest vulns impact mobile

By Mike Murray



2018



<https://blog.lookout.com/spectre-meltdown-vulnerabilities-mobile>

Operating System Layer

- Operating system: software running on a device that allows communications between the hardware and the application tiers
- Provides access to its resources via the publishing of application programming interfaces
- If an operating system flaw is discovered, the entire installed base of that particular operating system version will likely be vulnerable

Operating System Layer

iOS Trustjacking – A Dangerous New iOS Vulnerability

During our RSA Conference presentation today (Wednesday, April 16, 2018 | 9:15 AM PST | Moscone North 21), Adi Sharabani and myself disclosed a new iOS vulnerability which represents a new class of multi-device attacks

Intro

An iPhone user's worst nightmare is to have someone gain persistent control over his/her device, including the ability to record and control all activity without even needing to be in the same room. In this blog post, we present a new vulnerability called "Trustjacking", which allows an attacker to do exactly that.

This vulnerability exploits an iOS feature called iTunes Wi-Fi sync, which allows a user to manage their iOS device without physically connecting it to their computer. A single tap by the iOS device owner when the two are connected to the same network allows an attacker to gain permanent control over the device. In addition, we will walk through past related vulnerabilities and show the changes that Apple has made in order to mitigate them, and why these are not enough to prevent similar attacks.

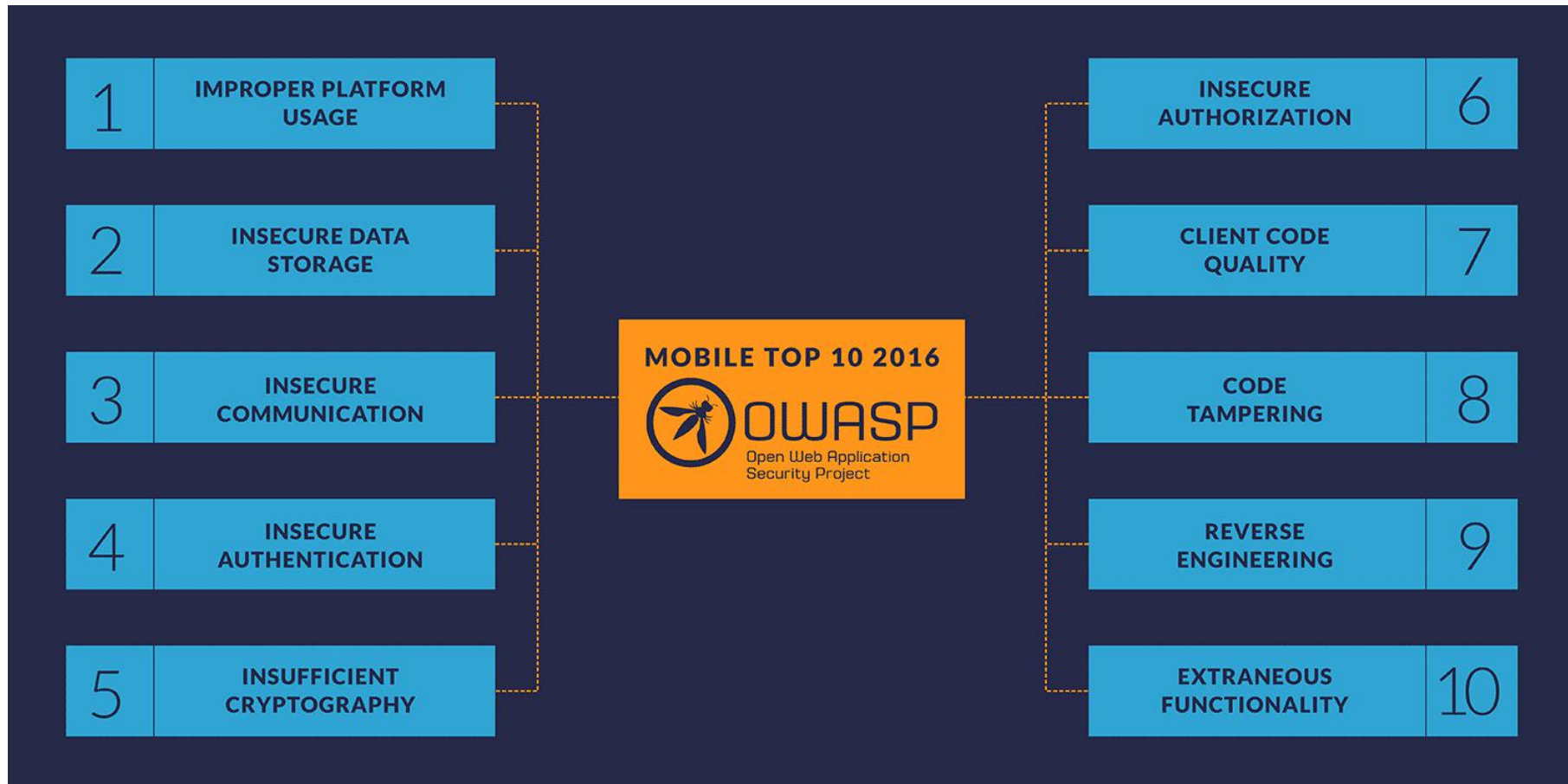
2018

<https://www.symantec.com/blogs/feature-stories/ios-trustjacking-dangerous-new-ios-vulnerability>

Application Layer

- Application: running processes that utilize application programming interfaces provided by the operating system layer as an entry point into the rest of the stack
- Typical flaws similar to computer application flaws. Examples:
 - buffer overflows
 - insecure storage of sensitive data
 - improper cryptographic algorithms
- The result of exploitation of application layer security flaws can range from elevated operating system privilege to exfiltration of sensitive data

Application Layer



https://www.owasp.org/index.php/Mobile_Top_10_2016-Top_10

Mobile OS security models

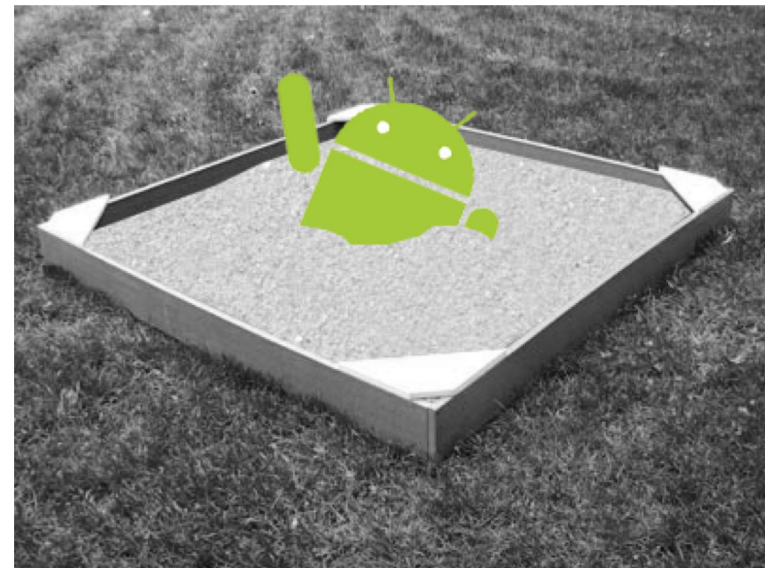


Android/iOS security model

– app execution sandbox

- Similar mechanism in both Android and iOS
- Each app is housed in a virtual sandbox

- Every app has access only to its own data and code, and as far as it knows, it's the only thing running on that device
- Apps are able to access photos and location only if the device owner gives permission
- An app can make its data available to other selected apps via well defined mechanisms



<https://www.slideshare.net/anushatuke1/android-sandbox>

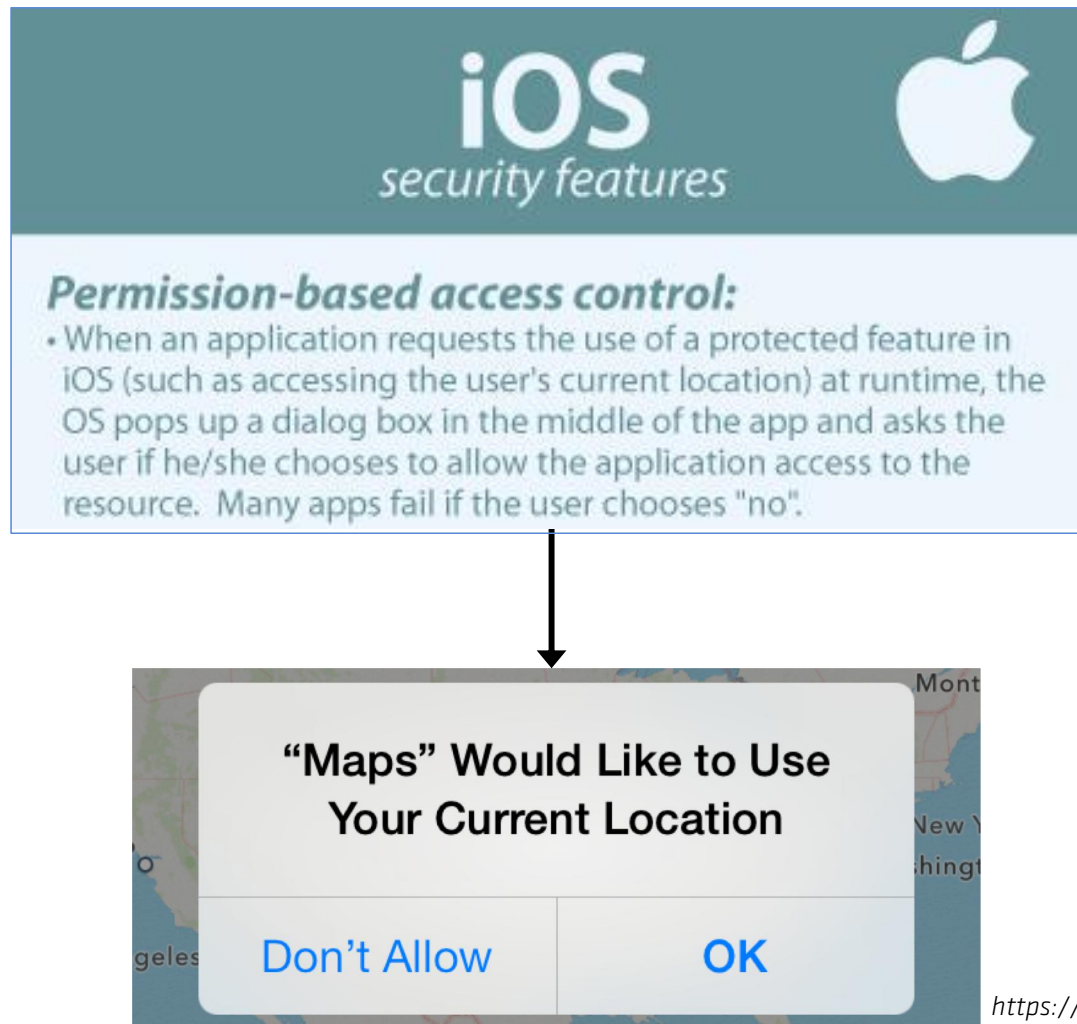
Android sandbox

- Apps written in C/Java
- Every app executes in its own VM
- Every app has its own unique Linux UID
 - Assigned by the OS at install time
- System apps have a larger sandbox
- Every app has a private directory, in which it has exclusive read/write privileges → */data/data/<appName>/*
- App data may be saved in a shared memory area, accessible by any other app → */sdcard/*

iOS sandbox

- Apps written in Objective-C/Swift
- Apps are ARM-compiled, no VM
- 2 system users:
 - Mobile: → all the apps execute with this UID
 - Root
- Systems apps do not execute inside a sandbox
- Non-system apps can only access their own private directory → */var/mobile/Containers/Data/Application/<appld>*

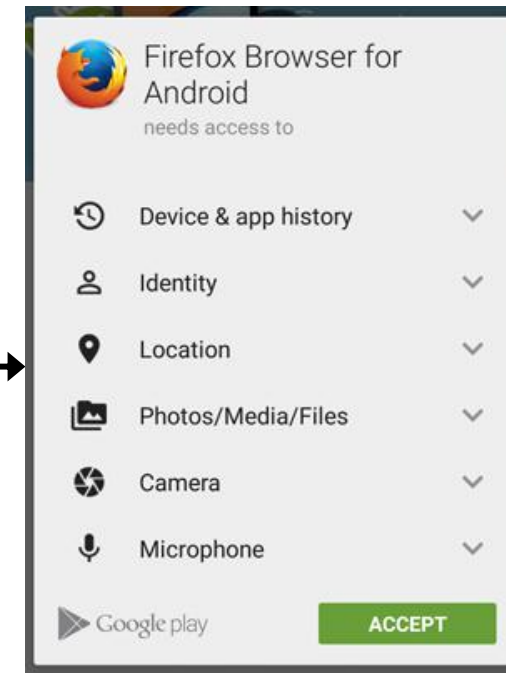
iOS – permissions



<https://www.veracode.com/resources/android-ios-security>

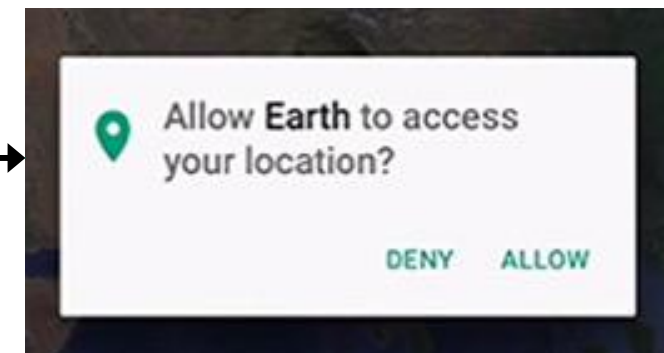
Android – permissions

Android ≤ 5



Android ≥ 6 (Marshmallow)

- Runtime permissions (like iOS)

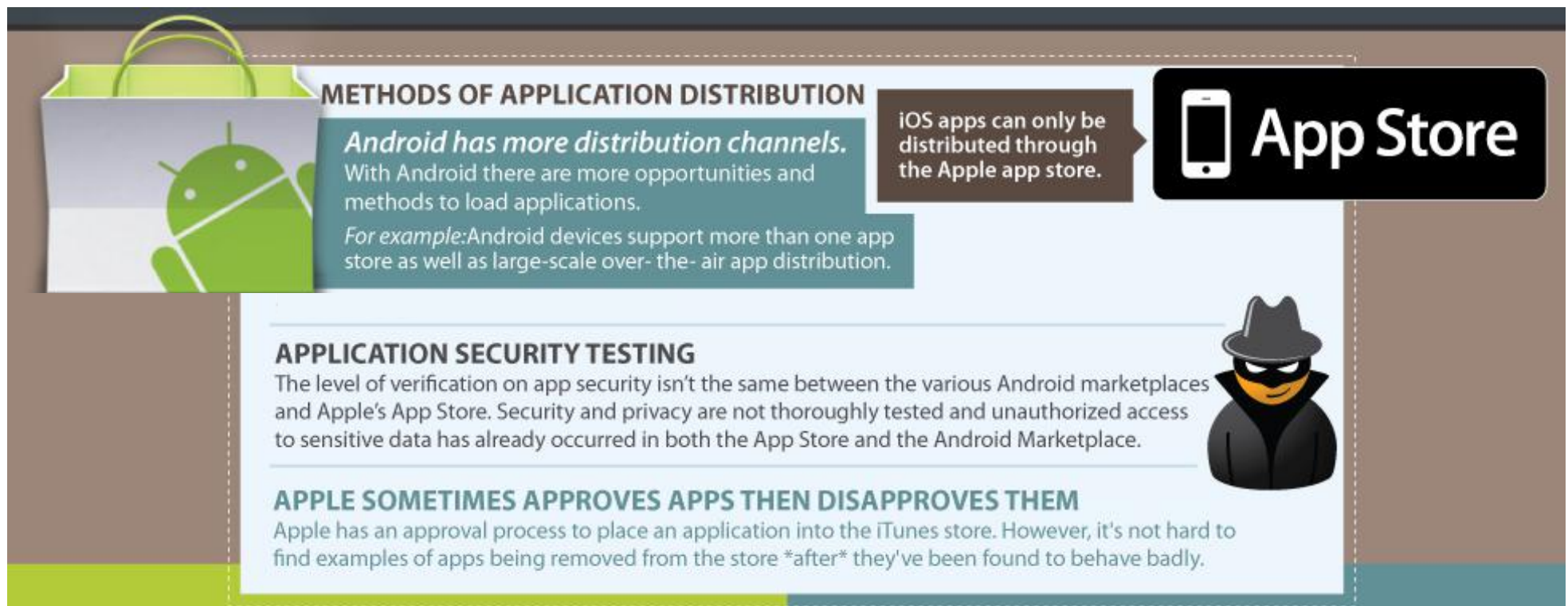


<https://www.veracode.com/resources/android-ios-security>

Android/iOS security model – application signing

- Only signed applications can execute
- Android:
 - Apps can be signed directly by the developer
- iOS:
 - Apps downloaded from the AppStore are signed only by Apple
 - Apps signed by a developer can execute if:
 - The user explicitly accepts it
 - Or the developer has an Enterprise Certificate

Android/iOS security model – application distribution



<https://www.veracode.com/resources/android-ios-security>

Android – Security-Enhanced Linux (SELinux)

- Since Android 4.3
- Mandatory Access Control (MAC) on top of Linux users



Limited capabilities even for root

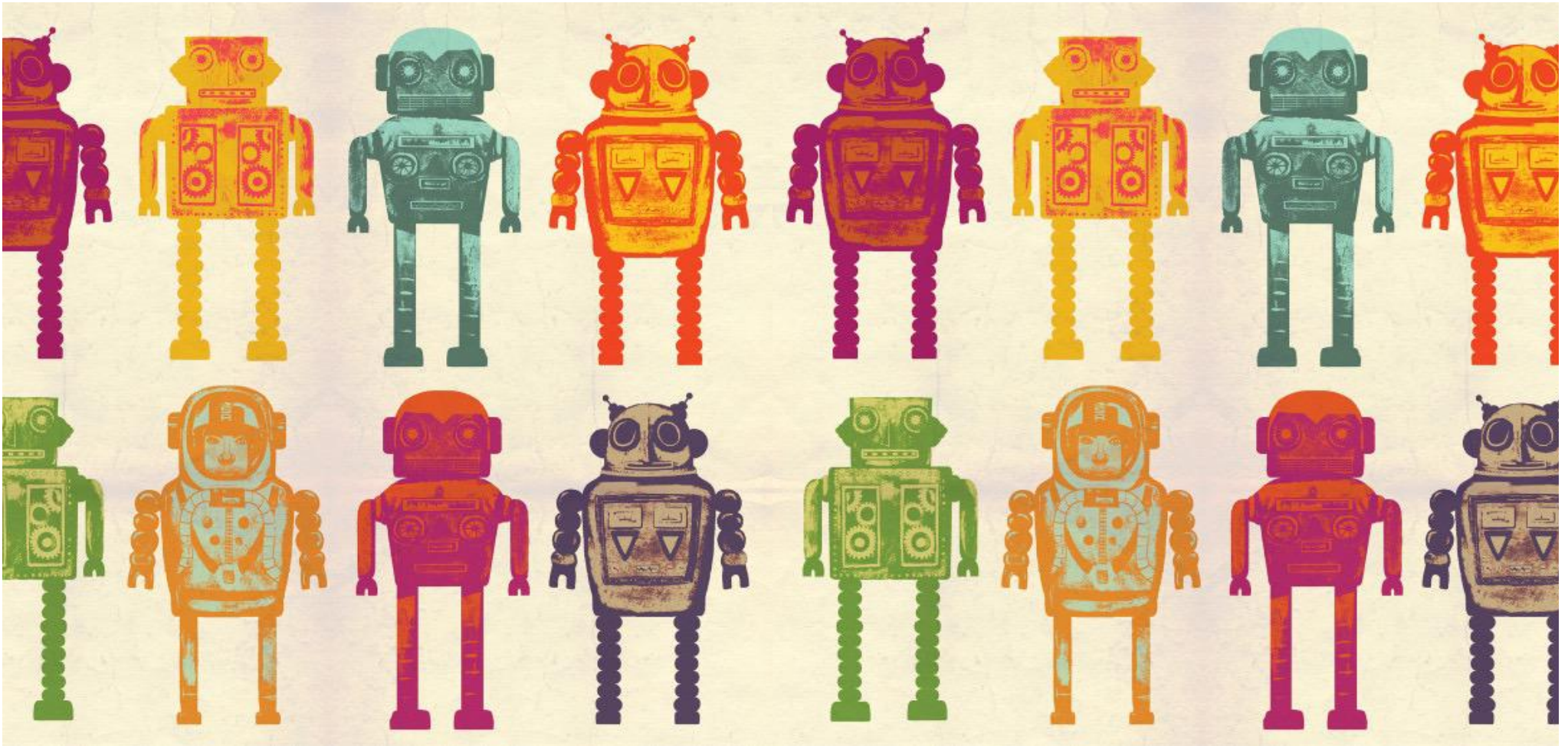


Limited capabilities even for malware that gains root privileges!



<https://selinuxproject.org/>

Mobile Malware



Mobile Malware

- Malicious software that targets mobile devices by causing the collapse of the system and loss or leakage of confidential information
- Who uses it:
 - Crime → Money
 - Government → Espionage
- Typical distribution → disguised as innocuous apps via:
 - App stores
 - Websites

Disguise


Sunny with a chance of stolen credentials: Malicious weather app found on Google Play

BY LUKAS STEFANKO POSTED 22 FEB 2017 - 03:00PM

MALWARE



Android users were the target of new banking malware with screen locking capabilities, which was disguised as a weather forecast app on Google Play.



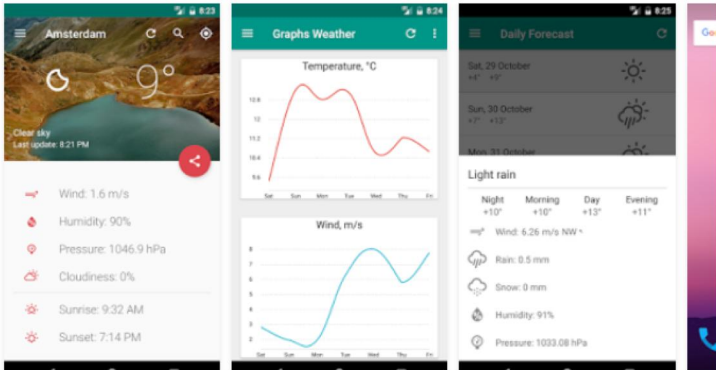
Good Weather

beauth ltd Weather ★★★★★ 30

PEGI 3

This app is compatible with your device.

Add to Wishlist Install



Good Weather: The most beautiful weather app. Ever.

Don't let bad weather take you by surprise! Set the gorgeous animated wallpapers with live weather conditions on your home screen and be aware of any weather that is coming your way. Whether it is cloudy, rainy, snowy or even stormy outside, Weather Live will provide you with current weather conditions and forecast in your city and multiple locations all around the world.

<https://www.welivesecurity.com/2017/02/22/sunny-chance-stolen-credentials-malicious-weather-app-found-google-play/>

Infection Methods

- Desktop typical infection method:
 - browser-based exploit
- Mobile infection methods:
 - WiFi Man in the Middle: the user connects to a malicious WiFi hotspot, which can try to intercept communications and install malicious apps
 - Distribution via application markets: distributing mass malware via application markets
 - Physical compromise: Hardly an occurrence on PCs, these are much more significant on the mobile

Sandbox escape

– Android LevelDropper (2016)

- Silently gains root access using publicly known exploits
- Installs other apps on the device

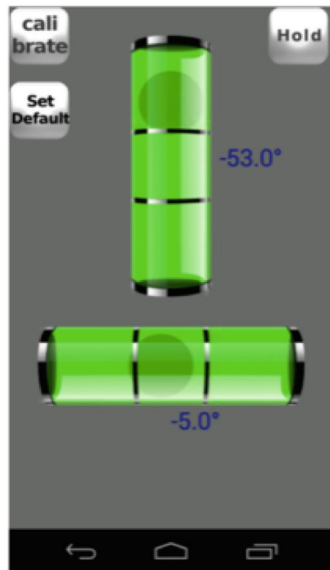


Figure 1. LevelDropper running

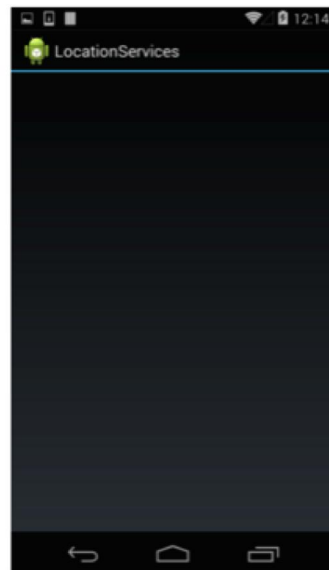


Figure 2. Shortly after opening, a blank LocationServices browser automatically opens

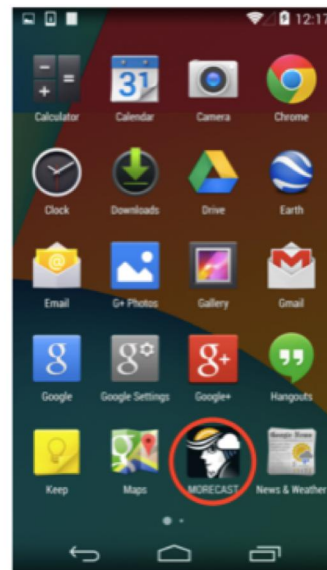


Figure 3. Closing LevelDropper, we see a new application installed



Figure 4. Without warning, this screen pops up. It appears to be the game installed without our knowledge

<https://blog.lookout.com/leveldropper>

Rooting Malware

[RU] <https://securelist.com/analysis/publications/74032/attack-on-zygote-a-new-twist-in-the-evolution-of-mobile-threats/>

LOG IN ▾

SECURELIST THREATS ▾ CATEGORIES ▾ TAGS ▾ ENCYCLOPEDIA ST

Rooting malware

2015

In our [previous article](#) we wrote about the increasing popularity of malware for Android that gains root access to a device and uses it to install apps and display aggressive advertising. Once this type of malicious program penetrates a device, it often becomes virtually impossible to use it due to the sheer number of annoying ads and installed apps.

»

Since the first article (August 2015), things have changed for the worse – the number of malware families of this type has increased from four to 11 and they are spreading more actively and becoming much better at “rooting”. According to our estimates, Trojans with superuser privileges attacked about 10% of Android-based mobile devices in the second half of 2015. There were also cases of these programs being pre-installed on new mobile devices coming from China.

Rooting Malware

Trends of the year

2017

Rooting malware: no surrender

For the last few years, rooting malware has been the biggest threat to Android users. These Trojans are difficult to detect, boast an array of capabilities, and have been very popular among cybercriminals. Their main goal is to show victims as many ads as possible and to silently install and launch the apps that are advertised. In some cases, the aggressive display of pop-up ads and delays in executing user commands can render a device unusable.

Rooting malware usually tries to gain **super-user rights** by exploiting system vulnerabilities that allow it to do almost anything. It installs modules in system folders, thus protecting them from removal. In some cases – Ztorg, for example –

<https://securelist.com/mobile-malware-review-2017/84139/>

Sandbox escape

– Android StageFright (2015)

- Attacker sends MMS with malicious code to victim's device
- When the device receives the MMS (before the user opens it):
 - The Hangouts app automatically process that video so it's ready for viewing as soon as you open the message
 - The malicious code runs instantaneously leveraging OS vulnerabilities in video processing libraries
 - Malware runs with privileged permissions!



<https://blog.zimperium.com/wp-content/uploads/2015/09/stagefrightExploit2.png>

Sandbox escape – iOS Pegasus (2016)

- Silently jailbreaks the device using zero-day exploits
- And then:
 - Reads text messages
 - Tracks calls
 - Collects passwords
 - Traces the phone location
 - Gathers information from apps including (but not limited to):
 - iMessage, Gmail, Viber, Facebook, WhatsApp, Telegram and Skype

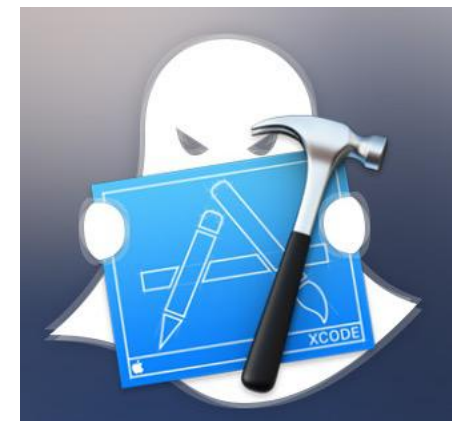


<https://blog.kaspersky.com/pegasus-spyware/14604/>
[https://en.wikipedia.org/wiki/Pegasus_\(spyware\)](https://en.wikipedia.org/wiki/Pegasus_(spyware))

Sandboxed malware

– iOS XcodeGhost (2015)

- Because of the slow download speed from Apple Servers, Chinese iOS developers used to download Xcode from third party websites
- Attackers distributed compromised Xcode versions on file hosting websites
- Malware embeds itself in every app compiled with the modified Xcode
- Malware collects user data and sends it to attacker's server



<http://www.redmondpie.com/apple-posts-xcodeghost-malware-faq-will-alert-users-who-installed-infected-apps/>

Spyware

- Spyware monitors user's activity, records location and lifts critical information, such as usernames and passwords for email accounts or e-commerce sites.
- In many cases, spyware is packaged with other seemingly benign software and quietly collects data in the background

Spyware – Android Skygofree (2018)

- Tracks the location of the device
- Turns on audio recording when the owner is in a certain place
- surreptitiously connects to a Wi-Fi network controlled by the attackers
- monitors popular apps such as Facebook Messenger, Skype, Viber, and WhatsApp

Skygofree — a Hollywood-style mobile spy

January 16, 2018

Most **Trojans** are basically the same: Having penetrated a device, they steal the owner's payment information, mine cryptocurrency for the attackers, or encrypt data and demand a ransom. But some display capabilities more reminiscent of Hollywood spy movies.



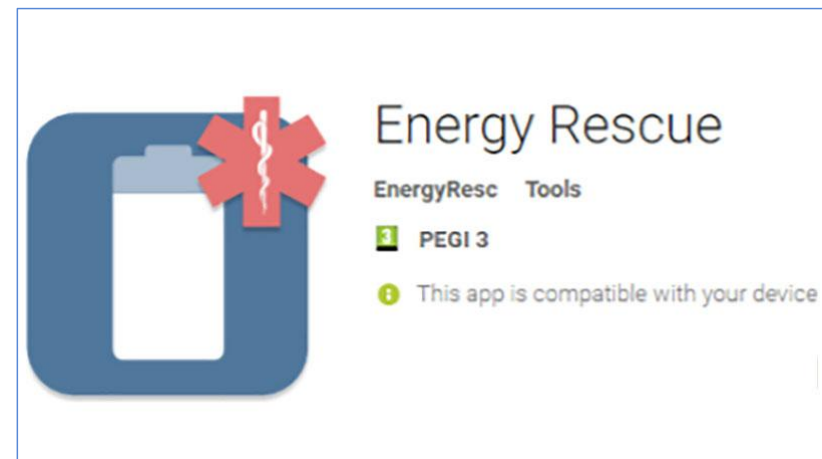
<https://www.kaspersky.com/blog/skygofree-smart-trojan/20717/>

Ransomware

- Ransomware "locks out" important user data such as documents, photos and videos typically by encrypting this information and then demanding a ransom be paid to the malware makers

Spyware + Ransomware = Energy Rescue (2017)

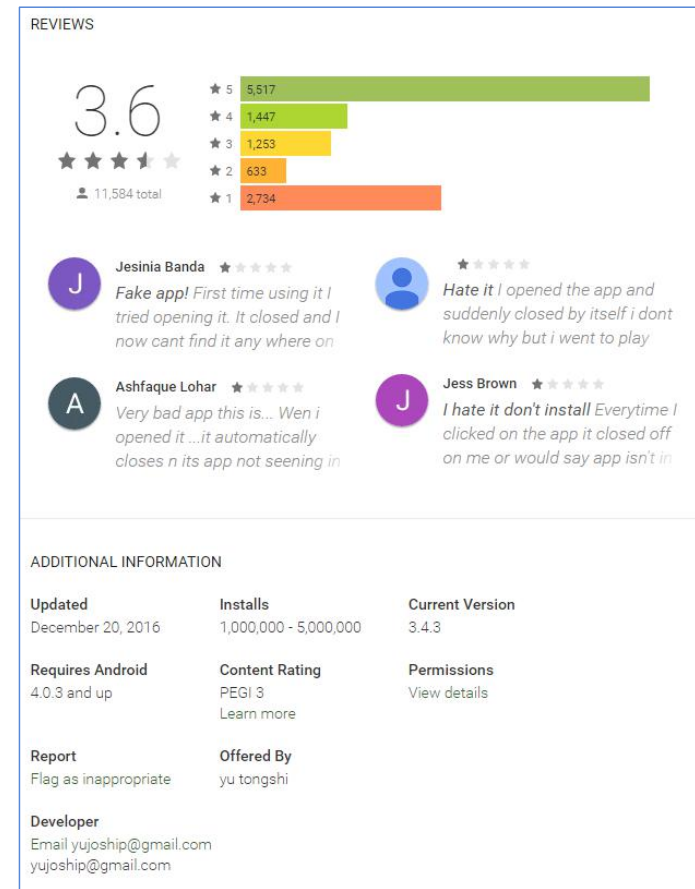
- Was downloadable on Google Play!
- Spyware:
 - Steals contacts
 - Steals SMS messages
- Ransomware
 - Asks the user to become a device administrator
 - User answers YES...
 - Malware locks the device and displays a message demanding payment



<http://blog.checkpoint.com/2017/01/24/charger-malware/>

Spyware + Ransomware = Energy Rescue (2017)

You need to pay for us, otherwise we will sell portion of your personal information on black market every 30 minutes. WE GIVE 100% GUARANTEE THAT ALL FILES WILL RESTORE AFTER WE RECEIVE PAYMENT. WE WILL UNLOCK THE MOBILE DEVICE AND DELETE ALL YOUR DATA FROM OUR SERVER! TURNING OFF YOUR PHONE IS MEANINGLESS, ALL YOUR DATA IS ALREADY STORED ON OUR SERVERS! WE STILL CAN SELLING IT FOR SPAM, FAKE, BANK CRIME etc... We collect and download all of your personal data. All information about your social networks, Bank accounts, Credit Cards. We collect all data about your friends and family.



<http://blog.checkpoint.com/2017/01/24/charger-malware/>

Adware

- Aggressive pop-ups and data collection
- "malvertising" code that can infect and root your device, forcing it to download specific adware types and allowing attackers to steal personal information

Adware – iOS Muda (2015)

- Designed for jailbroken devices
- Distributed via Cydia
- Once installed:
 - Displays advertisements over apps and in notification bar
 - Asks to user to install other promoted apps



<https://gist.github.com/secmobi/257166bb21d0a650fc93>

Adware – Android Judi (2017)

- Infected 36.5 million Android devices
- More than 41 malicious Android applications from a Korean company on Google Play Store
- adware programused to generate fraudulent clicks to generate revenue from advertisements



<https://thehackernews.com/2017/05/android-adware-malware.html>

SMS Trojan / WAP clicker

- SMS trojans wreak financial havoc by sending SMS messages to premium-rate numbers across the world, racking up users' phone bills
- WAP clickers open Web pages that have WAP billing and click buttons that initiate payments while the user suspects nothing

SMS Trojan / WAP clicker – Android Xafekopy (2017)



Virus

Android/Xafekopy.D!tr



Analysis

Android/Xafekopy.D!tr is classified as a trojan.

A trojan is a type of malware that performs activities without the user's knowledge. These activities commonly include establishing remote access connections, capturing keyboard input, collecting system information, downloading/uploading files, dropping other malware into the infected system, performing denial-of-service (DoS) attacks, and running/terminating processes.

The Fortinet Antivirus Analyst Team is constantly updating our descriptions. Please check the FortiGuard Encyclopedia regularly for updates.

<https://fortiguard.com/encyclopedia/virus/7425560>

Banking Trojan

- Banking trojans intercept text messages that include financial information and then send a copy of the text message through email or other means, giving cybercriminals all the information they need to infiltrate financial accounts

Banking Trojan

– Android Faketoken (2017)

- Impersonates the interfaces of taxi-booking apps
- Intercepts all phone calls
- Overlays legitimate app Uis with its own screen, asking for credit card data
- Intercepts all incoming SMS messages



<https://www.kaspersky.com/blog/faketoken-trojan-taxi/18002/>

Botnet Zombie

- Can gain complete access to the device and its contents
- Communicates with and receives instructions from one or more command and control servers (C&C)
- Every smartphone infected is added to a network of mobile bots (mobile botnet) managed by a cybercriminal called the botmaster
- Usage:
 - DDoS attacks
 - Advertising campaigns
 - ...

Botnet Zombie

– Android RottenSys (2018)

- Comes preinstalled with phones shipped from a particular chinese mobile devices distributor
- Comes initially with no malicious components and doesn't immediately start any malicious activity
- Communicate with its command-and-control servers to get the actual malicious code



<https://thehackernews.com/2018/03/android-botnet-malware.html>

Malware Analysis



(Mobile) malware analysis techniques



- Signature-based analysis
- Behavioral analysis
 - Static analysis
 - Dynamic analysis
 - Forensic analysis

Signature-based analysis

- How it works:
 - A new malware comes up somewhere
 - Analyst calculates a hash of the malware package and updates antivirus database
 - Antivirus is installed on the Google/Apple servers: every time an app is submitted by a developer it calculates the app package hash
 - Antivirus is installed on a device: every time an app is installed it calculates the app package hash
- Limitations:
 - Can't detect never-before-seen malware
 - Can't detect variations of known malware

Behavioral analysis

- Analyst defines what is a malicious behavior:
 - i.e.: an app may be malicious if it reads all the SMS messages and uploads them to a server
- App code analysis:
 - Analyst tries to understand what the app does
 - Analyst checks if what the app does follows a malicious behavior
- Pros:
 - Can detect never-before-seen malware and variations of known malware
- Cons:
 - May produce false positives

Behavioral analysis

Static Analysis

- App in a non-runtime context
- App re-engineering
- Static Code Review

Dynamic Analysis

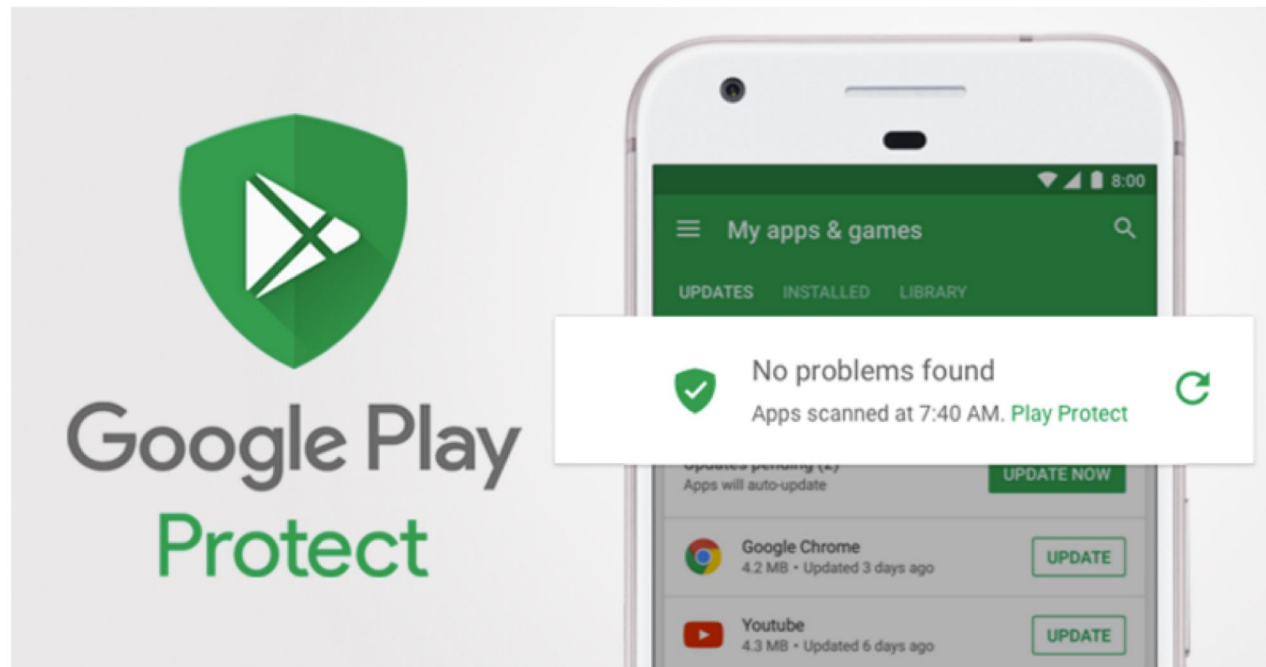
- Live network communications interception
- App in a runtime context
- Dynamic Code Review
- Live code mod

Behavioral analysis

Google Adds New Behavior-Based Malware Scanner To Every Android Device

Friday, May 19, 2017 Mohit Kumar

78 Like 1.7K Share 2594 Tweet 1245 Share 266 Share 4225



2017

<http://thehackernews.com/2017/05/google-play-protect-android.html>

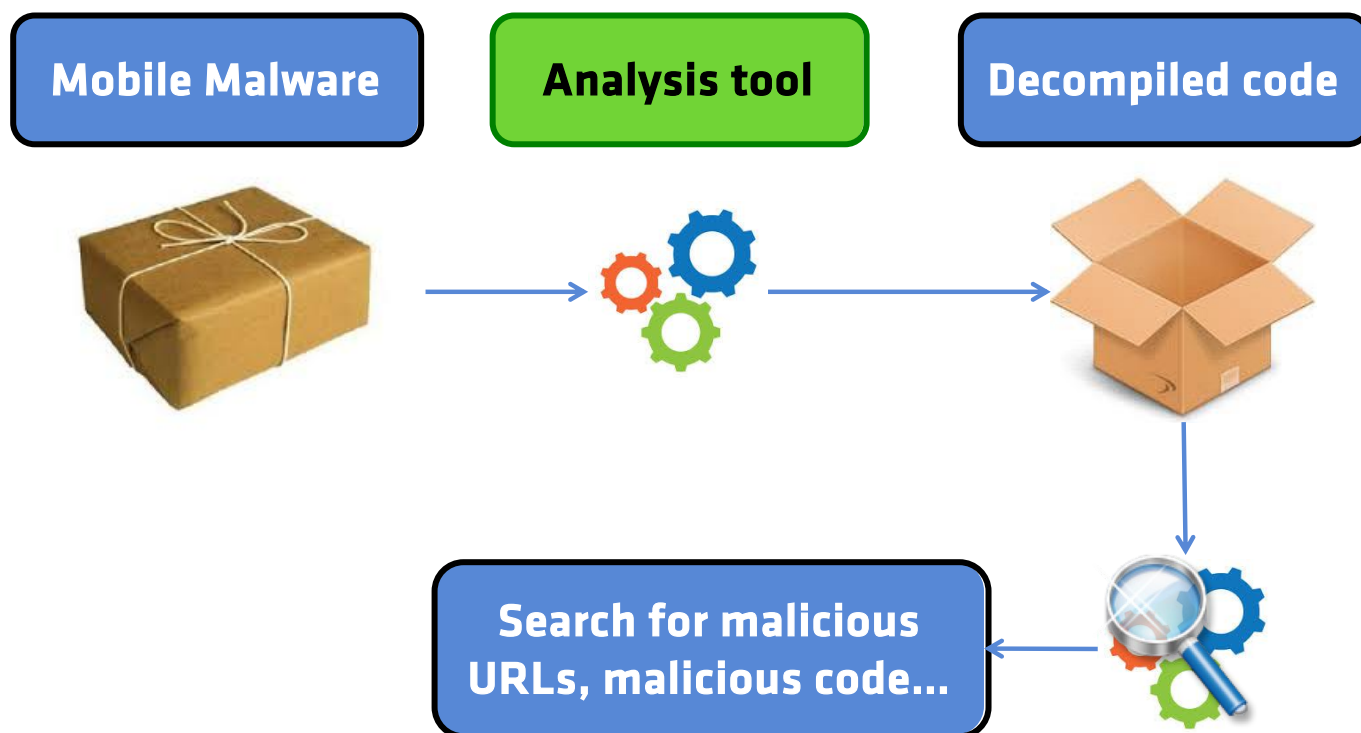
Behavioral analysis

Android		March 2018		
		Name	Protection	Usability
■ March 2018 (new)		AhnLab AhnLab V3 Mobile Security 3.1	●●●●●	●●●●●
■ January 2018		Alibaba Mobile Security 5.8	●●●●●	●●●●●
■ November 2017		Antiy AVL 2.5	●●●●●	●●●●●
■ September 2017		Avast Avast Mobile Security 6.9	●●●●●	●●●●●
■ July 2017		AVG AVG AntiVirus Free 6.9	●●●●●	●●●●●
■ May 2017		Avira Avira Antivirus Security Pro 5.2	●●●●●	●●●●●
■ March 2017		Bitdefender Bitdefender Mobile Security 3.2	●●●●●	●●●●●
■ January 2017		Cheetah Mobile Security Master 4.4	●●●●●	●●●●●
■ November 2016		F-Secure F-Secure Safe 17.2	●●●●●	●●●●●
■ September 2016		G Data Internet Security 26.2	●●●●●	●●●●●
■ July 2016		Google Google Play Protect 9.1	●●●●●	●●●●●
■ May 2016		Hi Security Hi Security Lab Hi Security 4.10	●●●●●	●●●●●
■ March 2016		Ikarus Ikarus mobile.security 1.7	●●●●●	●●●●●
■ January 2016		Kaspersky Lab Internet Security 11.15	●●●●●	●●●●●
■ November 2015		McAfee McAfee Mobile Security 4.9	●●●●●	●●●●●
■ September 2015		Norton Norton Norton Mobile Security 4.1	●●●●●	●●●●●
■ July 2015		PSafe PSafe DFNDR 5.4	●●●●●	●●●●●
■ May 2015		SOPHOS Sophos Mobile Security 7.2	●●●●●	●●●●●
■ March 2015		Tencent WeSecure 1.4	●●●●●	●●●●●
■ January 2015		Trend Micro Mobile Security & Antivirus 9.2	●●●●●	●●●●●

2018

<https://www.av-test.org/en/antivirus/mobile-devices/android/september-2017/>

Static Analysis



Disassemble/decompile code

Android Disassembly

ART bytecode → Smali code (Android "assembly")



Android Decompilation

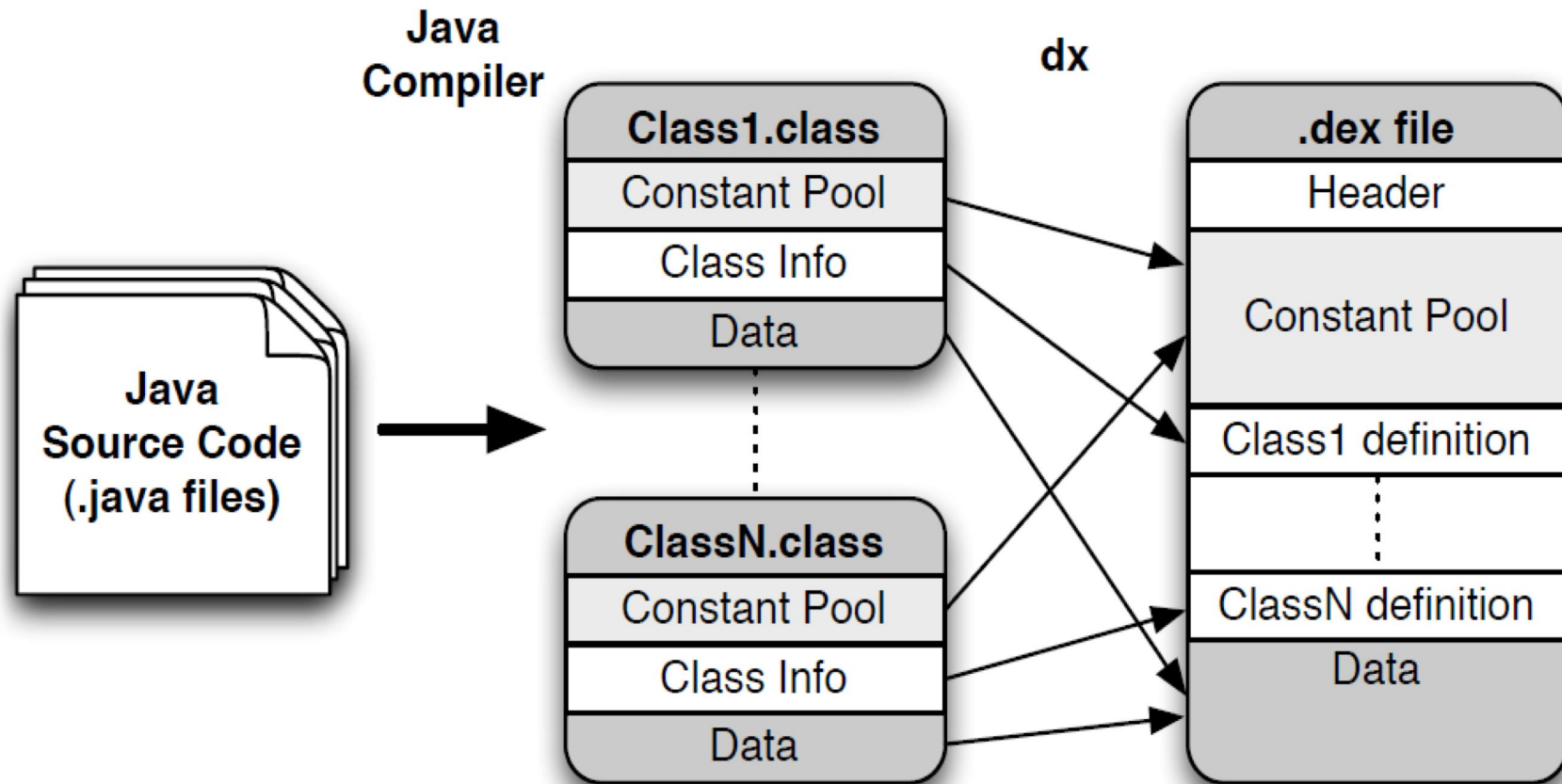
ART bytecode → JVM Bytecode → Java code

iOS Disassembly

ARM binary → Assembly → Pseudocode



Android: java → class → dex



Android disassembled ~~vs~~ ~~decompiled~~ code

```
 / ru trykov root RootDetection.smali

.class public Lru/trykov/root/RootDetection;
.super Lorg/apache/cordova/CordovaPlugin;
.source "RootDetection.java"

# direct methods
.method public constructor <init>()V
    .locals 0

    .prologue
    .line 17
    invoke-direct {p0}, Lorg/apache/cordova/CordovaPlugin;-><init>()V

    return-void
.end method

.method private checkBuildTags()Z
    .locals 2

    .prologue
    .line 37
    sget-object v0, Landroid/os/Build;->TAGS:Ljava/lang/String;

    .line 38
    .local v0, "buildTags":Ljava/lang/String;
    if-eqz v0, :cond_0
```

ApkTool:
<https://ibotpeaches.github.io/Apktool/>

Android ~~disassembled~~ vs decompiled code

```

/ ru trykov root RootDetection.java

package ru.trykov.root;

import android.os.Build;
import java.io.File;
import org.apache.cordova.CallbackContext;
import org.apache.cordova.CordovaPlugin;
import org.json.JSONArray;
import org.json.JSONException;

public class RootDetection
extends CordovaPlugin {
    private boolean checkBuildTags() {
        String string2 = Build.TAGS;
        if (string2 != null && string2.contains("test-keys")) {
            return true;
        }
        return false;
    }

    private boolean checkFilePath() {
        String[] arrstring = new String[]{"sbin/su", "/system/bin/su", "/system/xbin/su", "/data/local/xbin/su", "/data
n/failsafe/su", "/data/local/su"};
        int n2 = arrstring.length;
        for (int i2 = 0; i2 < n2; ++i2) {
            if (!new File(arrstring[i2]).exists()) continue;
            return true;
        }
        return false;
    }
}
```

Dex2jar: <https://github.com/pxb1988/dex2jar>

Lots of java decompilers: <http://www.javadecompilers.com/>

iOS Objective-C headers

class-dump: <https://github.com/nygard/class-dump>

/

jailbreakDetection.h

```
@interface JailbreakDetection : CDVPlugin
{
}

- (_Bool)jailbroken;
- (void)isJailbroken:(id)arg1;

@end
```

Class name and
inheritance

Methods
signatures

iOS disassembled code vs ~~decompiled pseudocode~~

Hopper:
<https://www.hopperapp.com/>

Hopper:
<https://www.hopperapp.com/>

```

; ===== BEGINNING OF PROCEDURE

; Variables:
;   var_1C: -28
;   var_20: -32
;   var_24: -36

-[JailbreakDetection jailbroken]:
00038a08    push    {r4, r5, r6, r7, lr}
00038a0a    add     r7, sp, #0xc
00038a0c    push.w  {r8, sl, fp}
00038a10    sub     sp, #0xc
00038a12    movw    r0, #0x9422
00038a16    movt    r0, #0x39
00038a1a    movw    r8, #0xa9d0
00038a1e    movt    r8, #0x39
00038a22    add     r0, pc
00038a24    add     r8, pc
00038a26    ldr.w   sl, [r0]
00038a2a    ldr.w   r0, [r8]
00038a2e    mov     r1, sl
00038a30    blx     imp__picsymbolstub4__objc_msgSend
00038a34    mov     r7, r7
00038a36    blx     imp__picsymbolstub4__objc_retainAutoreleasedReturnValue
00038a3a    mov     r6, r0
00038a3c    movw    r0, #0x9548
00038a40    movt    r0, #0x39
00038a44    movw    r2, #0xa5d2
; Objective C Implementation defined at 0x3c8
; @selector(defaultManager), :lower16:(0x3d1e
; @selector(defaultManager), :upper16:(0x3d1e
; :lower16:(0x3d33f8 - 0x38a28)
; :upper16:(0x3d33f8 - 0x38a28)
; @selector(defaultManager)
; objc_cls_ref_NSFileManager
; "defaultManager",@selector(defaultManager)
; objc_cls_ref_NSFileManager,_OBJC_CLASS_$_NS
; @selector(fileExistsAtPath:), :lower16:(0x3
; @selector(fileExistsAtPath:), :upper16:(0x3
; @"/Applications/Cydia.app", :lower16:(0x3c3

```

iOS ~~disassembled code~~ vs decompiled pseudocode

```
#import "JailbreakDetection.h"
```

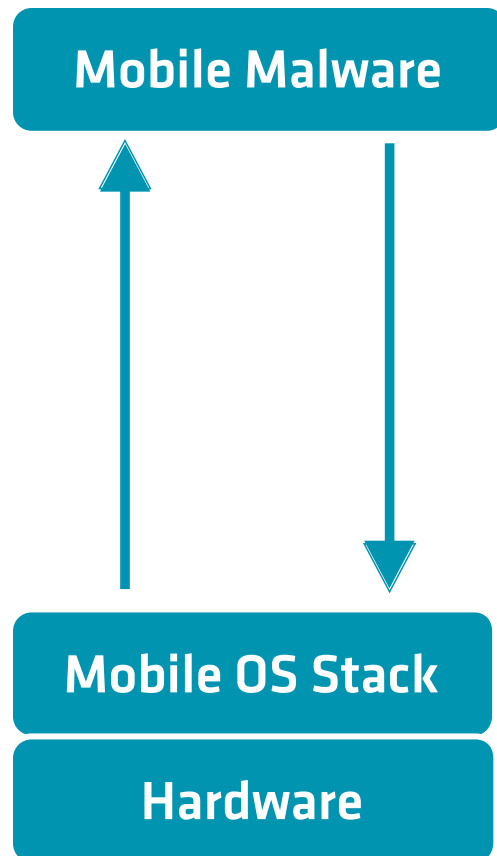
```
@implementation JailbreakDetection
```

```
- (_Bool)jailbroken {  
    r7 = (sp - 0x14) + 0xc;  
    sp = sp - 0x2c;  
    r7 = r7;  
    r6 = [[NSFileManager defaultManager] retain];  
    r4 = [r6 fileExistsAtPath:@"Applications/Cydia.app"];  
    [r6 release];  
    if ((r4 & 0xff) == 0x0) {  
        r0 = [NSFileManager defaultManager];  
        r7 = r7;  
        r0 = [r0 retain];  
        r6 = [r0 fileExistsAtPath:@"Library/MobileSubstrate/MobileSubstrate.dylib"];  
        [r0 release];  
        if ((r6 & 0xff) == 0x0) {  
            r0 = [NSFileManager defaultManager];  
            r7 = r7;  
            r0 = [r0 retain];  
            r6 = [r0 fileExistsAtPath:@"bin/bash"];  
        }  
    }  
}
```

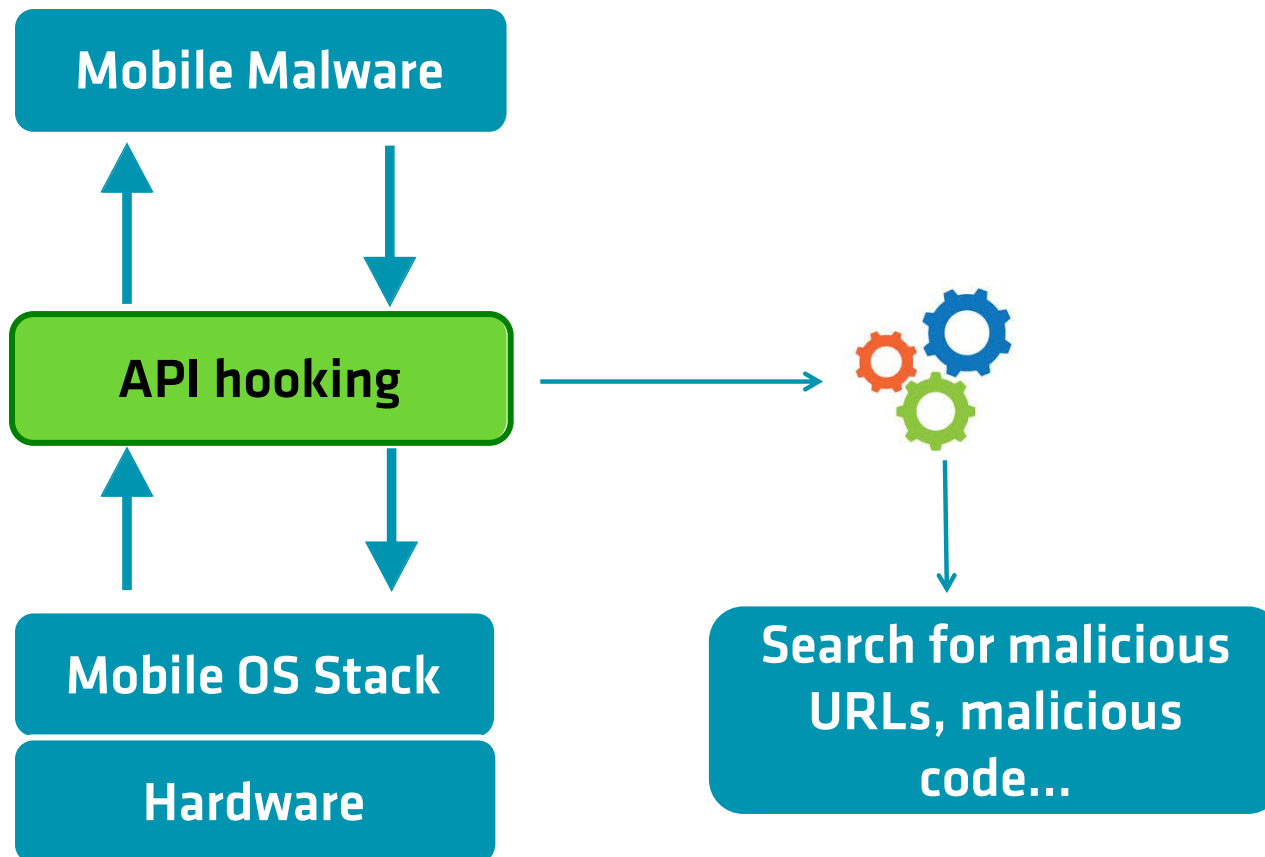
Hopper:

<https://www.hopperapp.com/>

Dynamic Analysis (malware – mobile OS)



Dynamic Analysis (malware – mobile OS)



Dynamic Analysis (malware – internet)

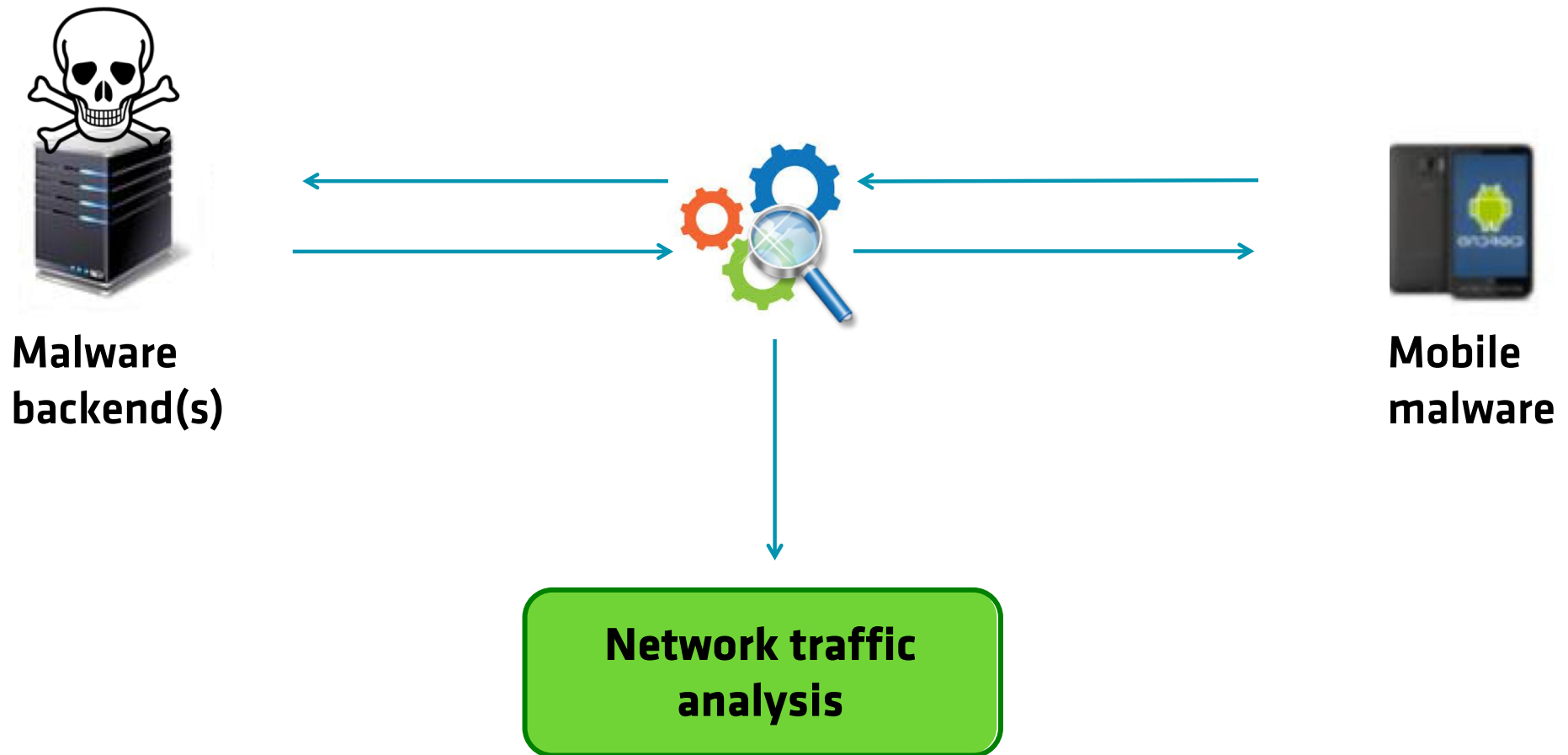


**Malware
backend(s)**



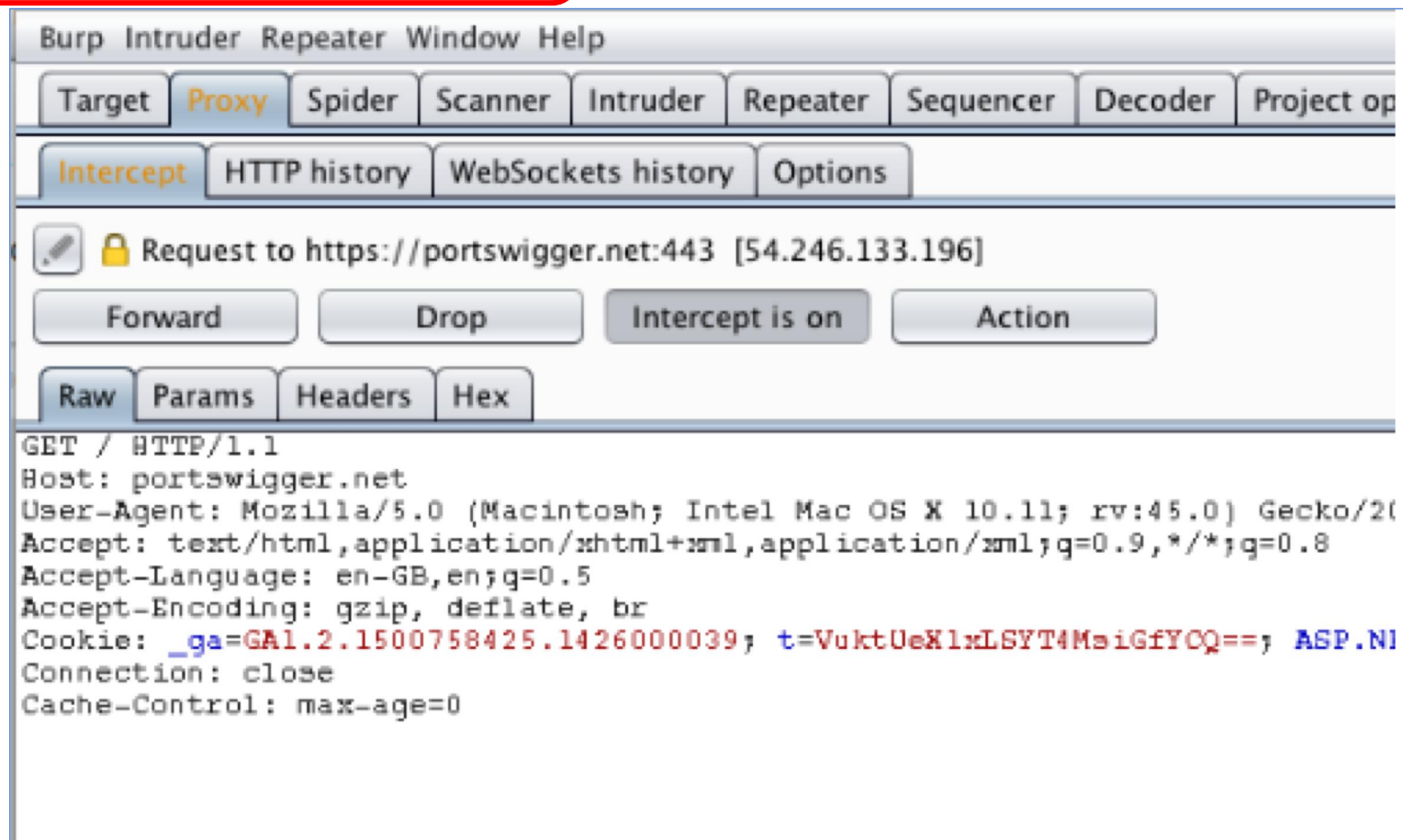
**Mobile
malware**

Dynamic Analysis (malware – internet)

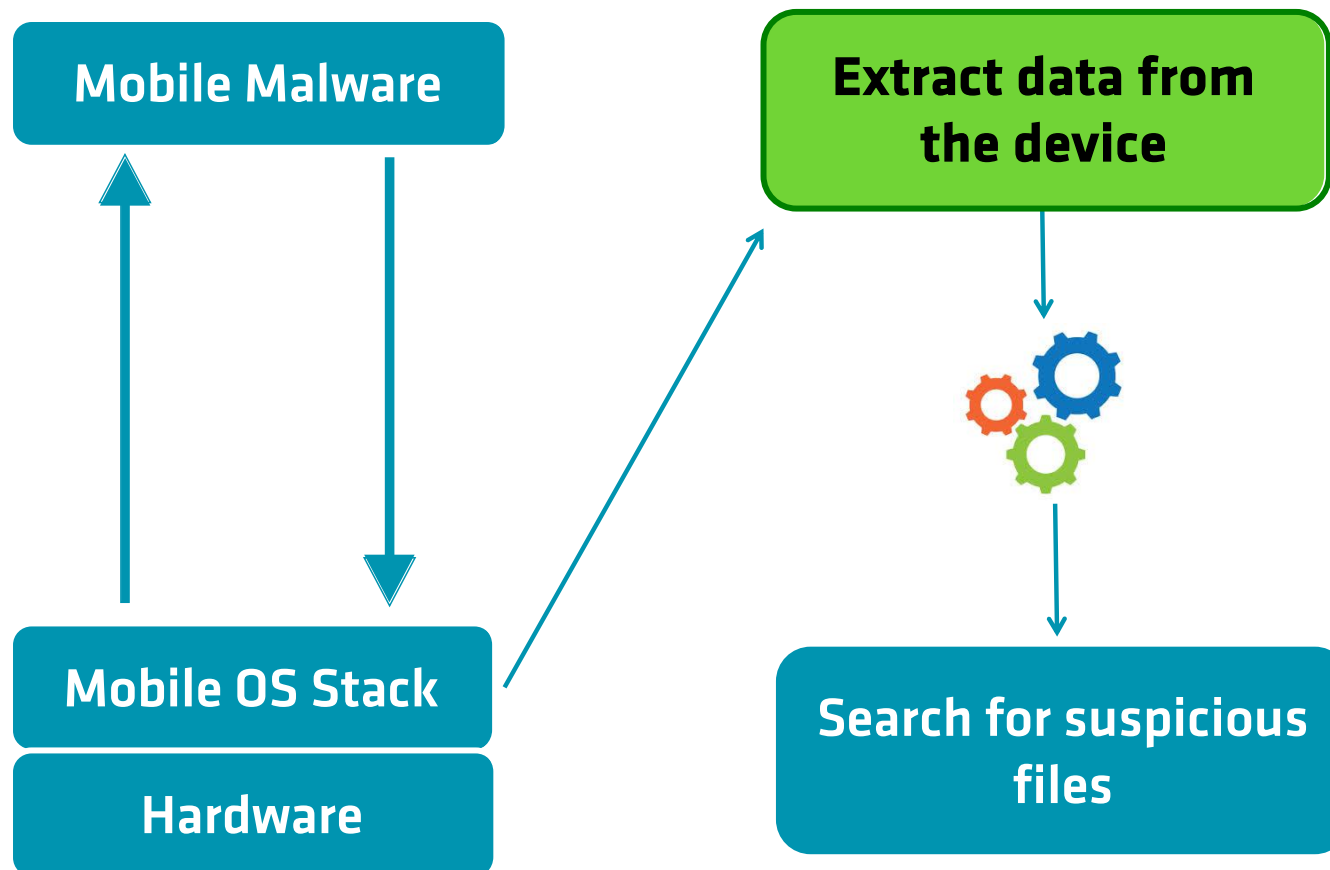


Dynamic Analysis (malware – internet)

Burp Suite:
<https://portswigger.net/>



Forensic Analysis (malware – filesystem)



Forensic Analysis (malware – filesystem)

- Timeline analysis
 - Analyze timestamps created from in file system
- Analysis of different file types
 - SQLite databases (Android & iOS)
 - Log files (Android & iOS)
 - Cookies (Android & iOS)
 - Screenshots (iOS)
 - Keyboard cache (iOS)
 - SharedPreferences (Android)
 - Keychain (iOS)

https://www.owasp.org/images/0/04/Security_Testing_Guidelines_for_mobile_Apps_-_Florian_Stahl%2BJohannes_Stroeher.pdf

Forensic Analysis (malware – filesystem)

fsmon: <https://github.com/nowsecure/fsmon>

```
root@hammerhead:/ # /data/local/tmp/inotifywait -r -m /data/data
Setting up watches. Beware: since -r was given, this may take a while!
Watches established.
/data/data/com.google.android.gms/databases/ OPEN playlog.db
/data/data/com.google.android.gms/databases/ ACCESS playlog.db
/data/data/com.google.android.gms/databases/ CREATE playlog.db-wal
/data/data/com.google.android.gms/databases/ OPEN playlog.db-wal
/data/data/com.google.android.gms/databases/ ATTRIB playlog.db-wal
/data/data/com.google.android.gms/databases/ CREATE playlog.db-shm
/data/data/com.google.android.gms/databases/ OPEN playlog.db-shm
/data/data/com.google.android.gms/databases/ ATTRIB playlog.db-shm
/data/data/com.google.android.gms/databases/ MODIFY playlog.db-shm
/data/data/com.google.android.gms/databases/ MODIFY playlog.db-shm
/data/data/com.google.android.gms/databases/ ACCESS playlog.db
/data/data/com.google.android.gms/databases/ ATTRIB playlog.db
/data/data/com.google.android.gms/databases/ OPEN playlog.db
/data/data/com.google.android.gms/databases/ ACCESS playlog.db
/data/data/com.google.android.gms/databases/ OPEN playlog.db-wal
/data/data/com.google.android.gms/databases/ ACCESS playlog.db
```

<https://www.nowsecure.com/blog/2016/02/18/filesystem-monitor-tool-for-ios-and-android/>

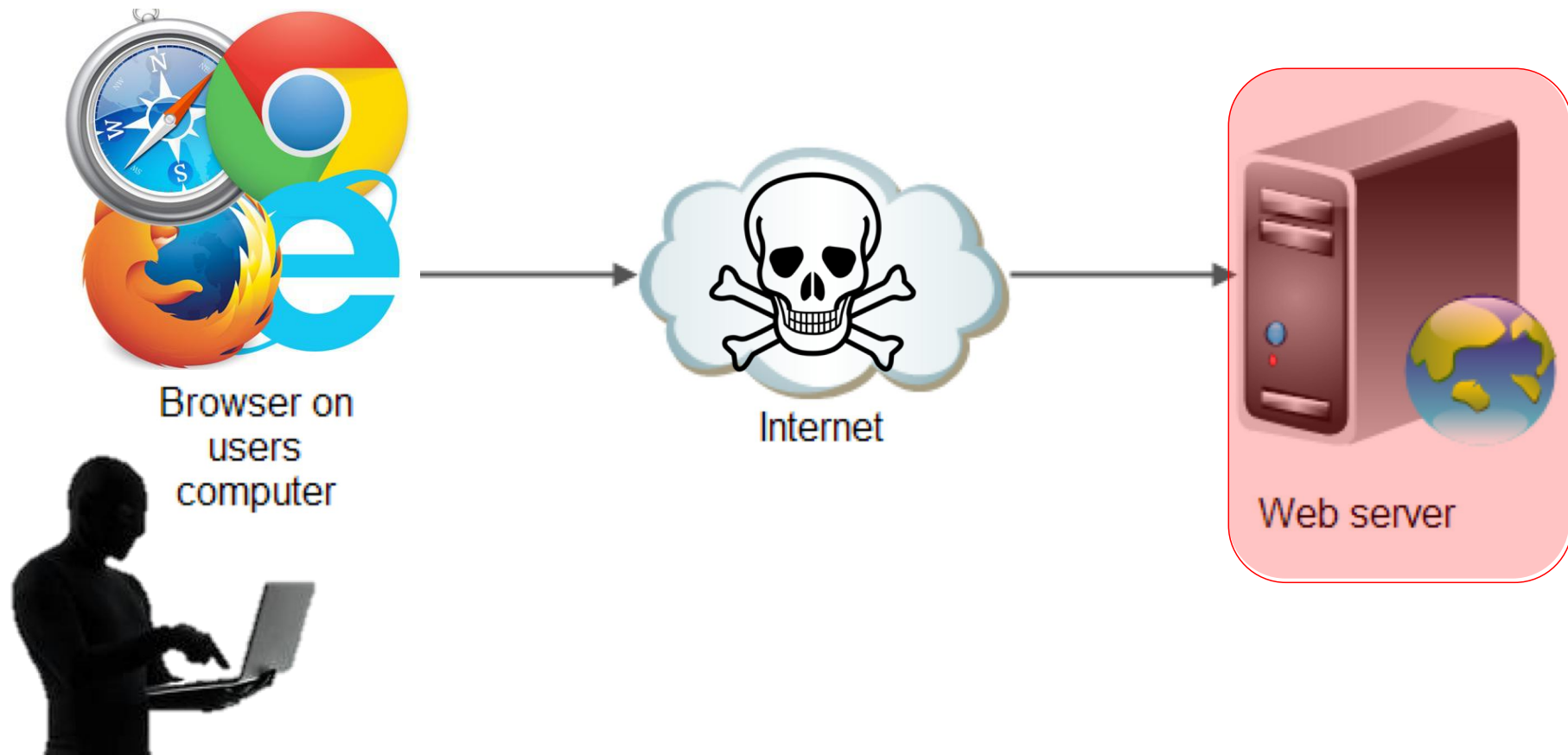
Part 2

Mobile app security

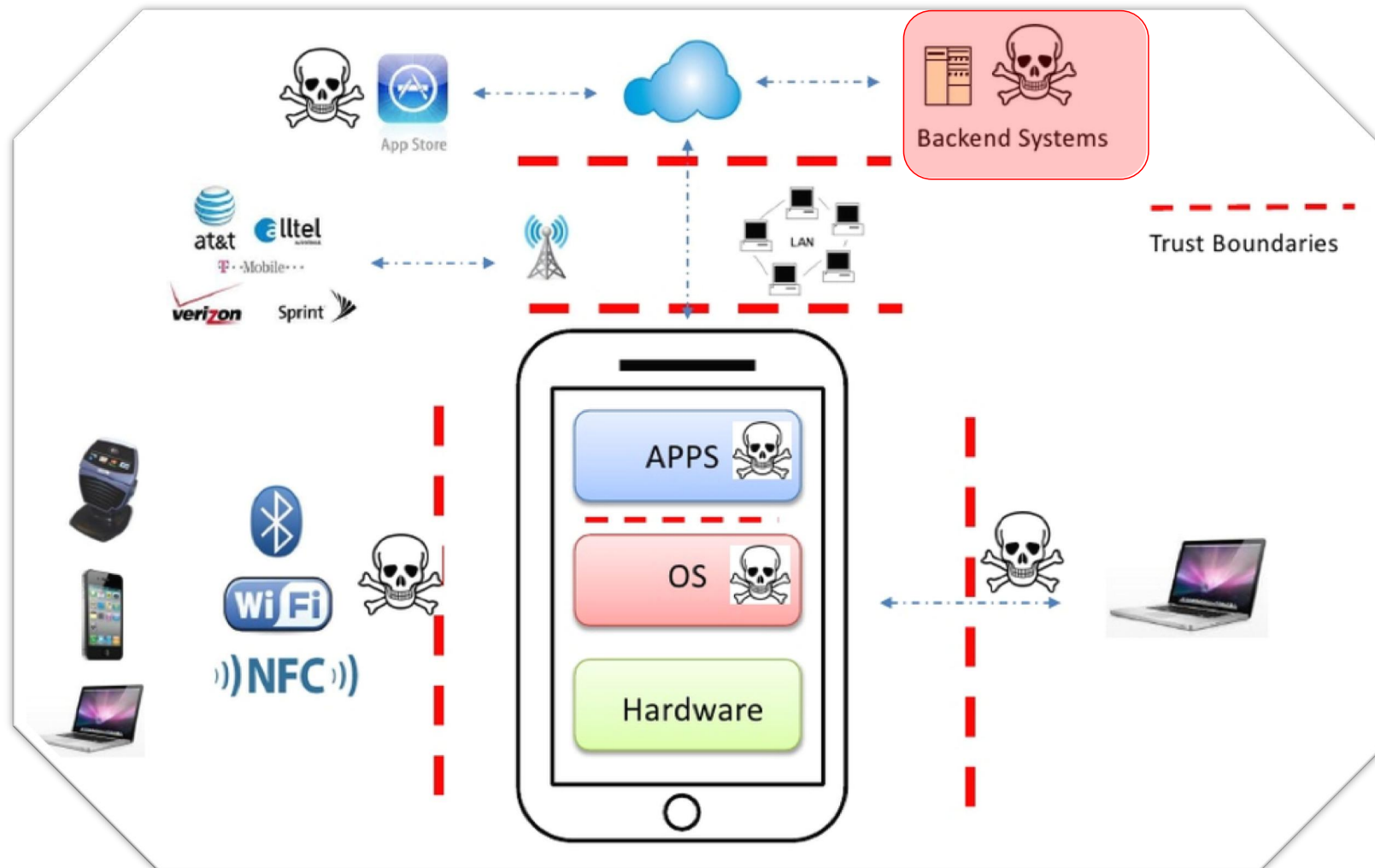
Mobile App Analysis: what and why

- By 2021 estimated devices per capita are expected to be 1.5 and mobile data traffic will reach the amount of 49 exabytes
- Mobile Apps are pervasive in our life supporting us from simple action, such as photo sharing, to more important actions, such as banking transactions.
- Security around these operations and data is crucial, making App vulnerability analysis and code review fundamental.

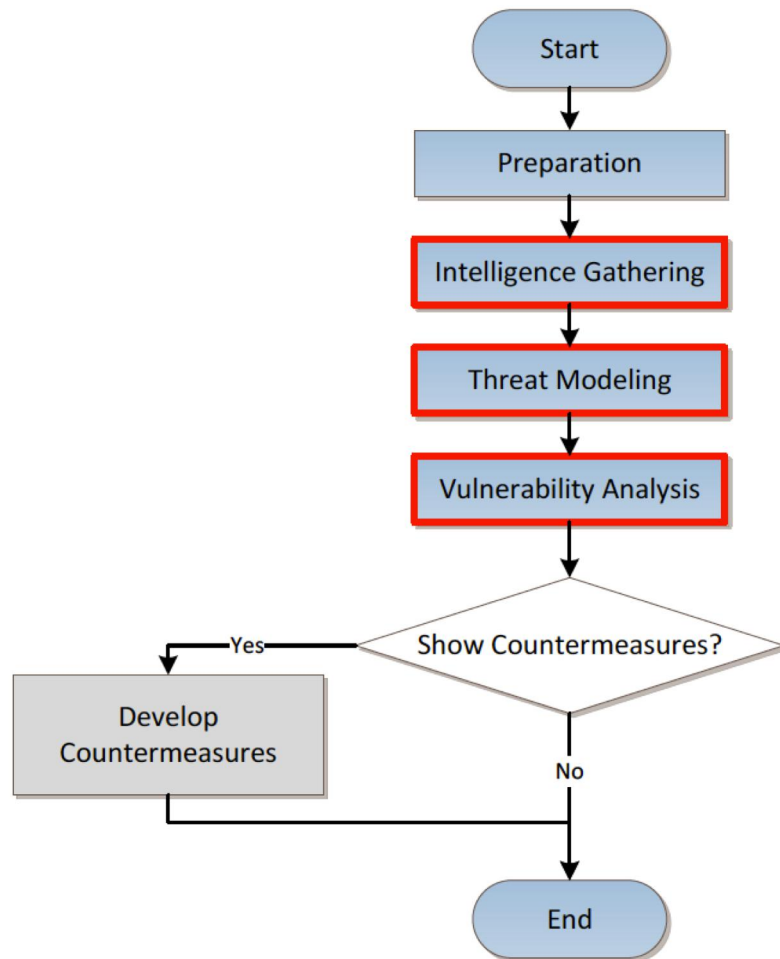
Web App Threats



Mobile App Threats



Security Testing Overview



The specific sub-processes are different for Android and iOS

Explanation:

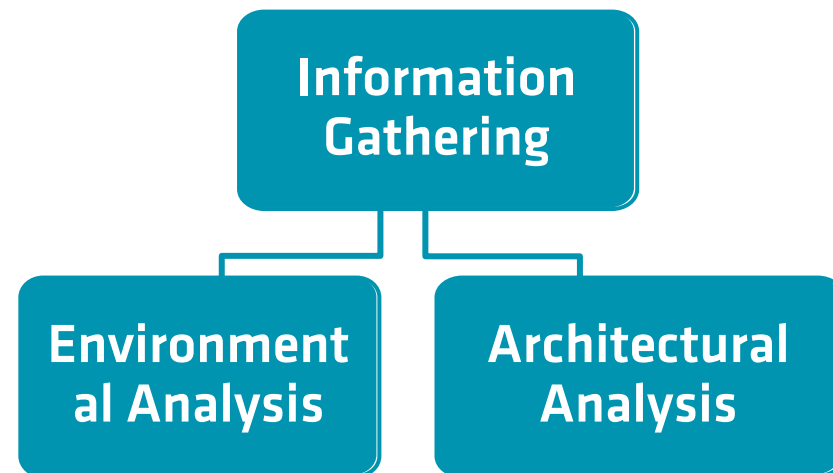
Mandatory

Optional

https://www.owasp.org/images/0/04/Security_Testing_Guidelines_for_mobile_Apps_-_Florian_Stahl%2BJohannes_Stroeher.pdf

Intelligence Gathering

- Try to catch as much as possible information about the app
- Consists of 2 analysis
 - Environmental
 - Architectural



- Consider mobile specific requirements

https://www.owasp.org/images/0/04/Security_Testing_Guidelines_for_mobile_Apps_-_Florian_Stahl%2BJohannes_Stroeher.pdf

Intelligence Gathering

- Environmental Analysis
 - Focus on the company behind the app and their business case and the relating stakeholders
 - Analyze internal processes and structures
- Architectural Analysis
 - App (network interfaces, used data, communication with other resources, session management, jailbreak/rooting detection, ...)
 - Runtime environment (MDM, jailbreak/rooting, os version)
 - Backend services (application server, databases, firewall, ...)

https://www.owasp.org/images/0/04/Security_Testing_Guidelines_for_mobile_Apps_-_Florian_Stahl%2BJohannes_Stroeher.pdf

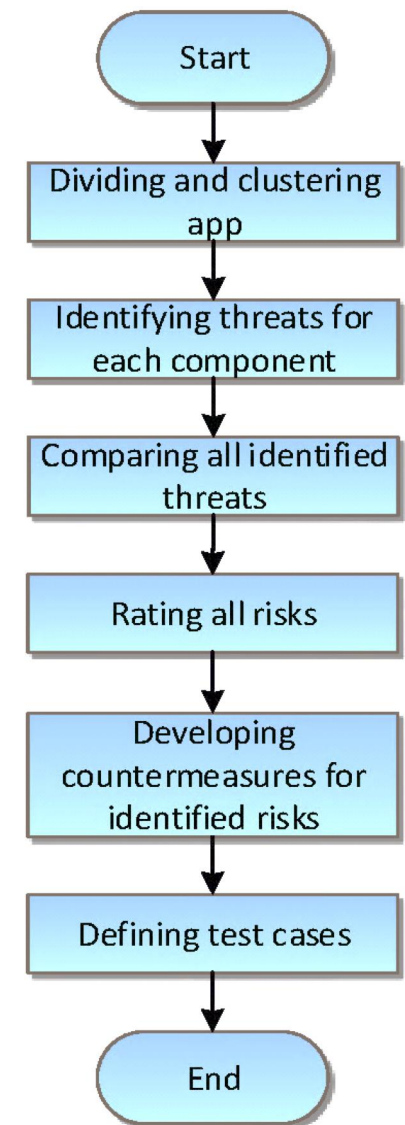
Intelligence Gathering – Example

- Examples for collected information from the Architectural Analysis for an example app:
 - User session remains until the user logs off manually
 - Financial transactions are included
 - Runs on a jailbroken device → no jailbreak detection
 - Provides operations on server side data for creating, reading, updating, deleting user data, items, shipping information...
- Backend services
 - Details about the version of the running service (Apache, Wordpress...)

https://www.owasp.org/images/0/04/Security_Testing_Guidelines_for_mobile_Apps_-_Florian_Stahl%2BJohannes_Stroeher.pdf

Threat Modeling

- Identifying threats for the app - specific or prepared threats (e.g. OWASP Top 10)
 - Should be done already in the development
- Risk rating (e.g. with OWASP Risk Rating)
- Developing countermeasures (e.g. with best practices or developers guides)
- Threat Modeling makes the complete process more traceable and efficient for all participants



https://www.owasp.org/images/0/04/Security_Testing_Guidelines_for_mobile_Apps_-_Florian_Stahl%2BJohannes_Stroeher.pdf

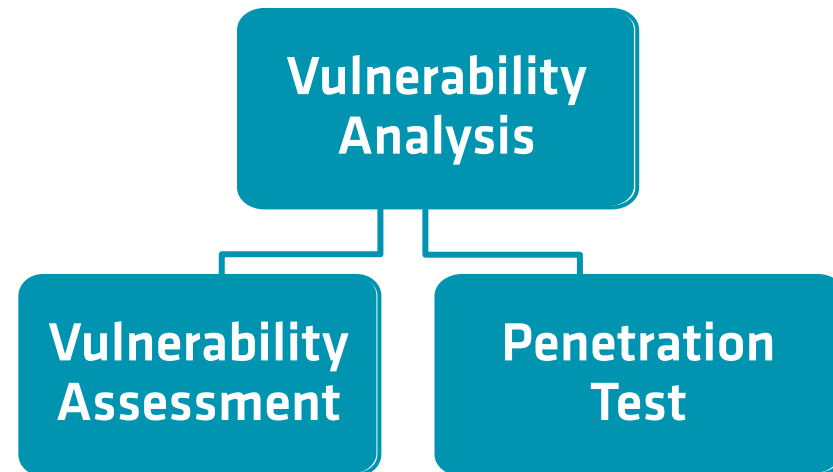
Threat Modeling – Example

- Information from the Intelligence Gathering
 - App provides operations on server side data
- Specific threat
 - Unauthorized reading of data on the network traffic while communicating with the backend
- Relating countermeasure
 - Implementing a secure transport layer protection (e. g. SSL, TLS)
- Relating test case
 - Try to catch and read the network traffic between the App and the backend

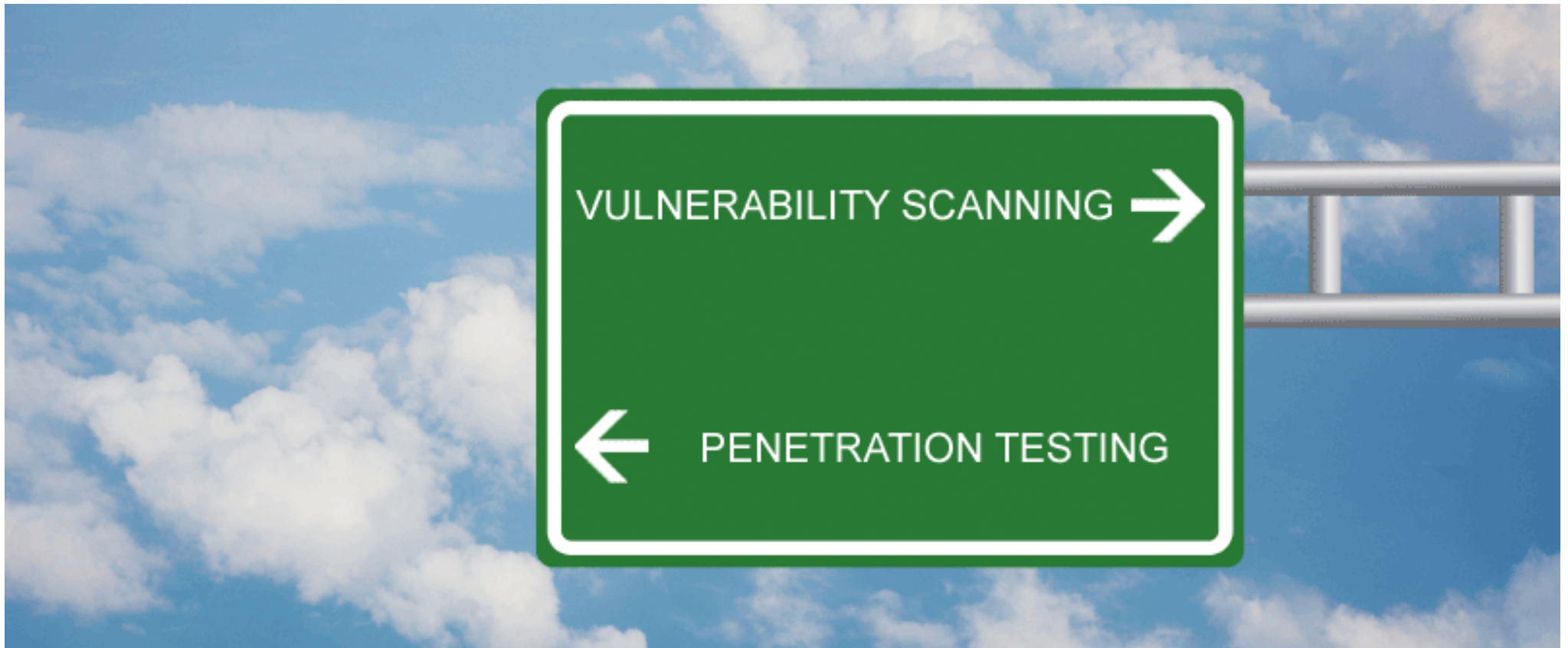
https://www.owasp.org/images/0/04/Security_Testing_Guidelines_for_mobile_Apps_-_Florian_Stahl%2BJohannes_Stroeher.pdf

Vulnerability Analysis

- Identifying vulnerabilities in the app with the previously created test cases
- Executing test cases with techniques from 2 different categories:
 - Vulnerability Assessment
 - Penetration Test



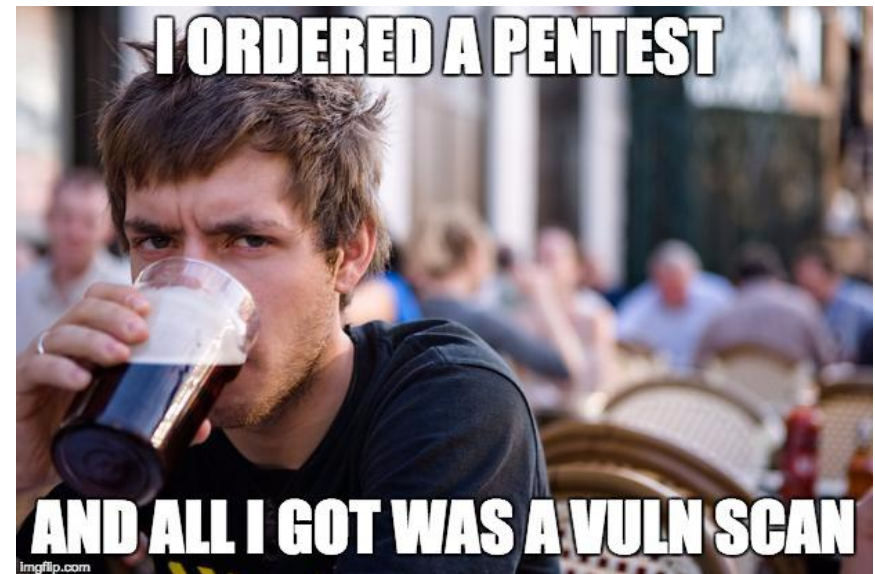
Vulnerability Assessment & Penetration Test



https://www.htbridge.com/blog/how_to_keep_your_website_safe_in_2015.html

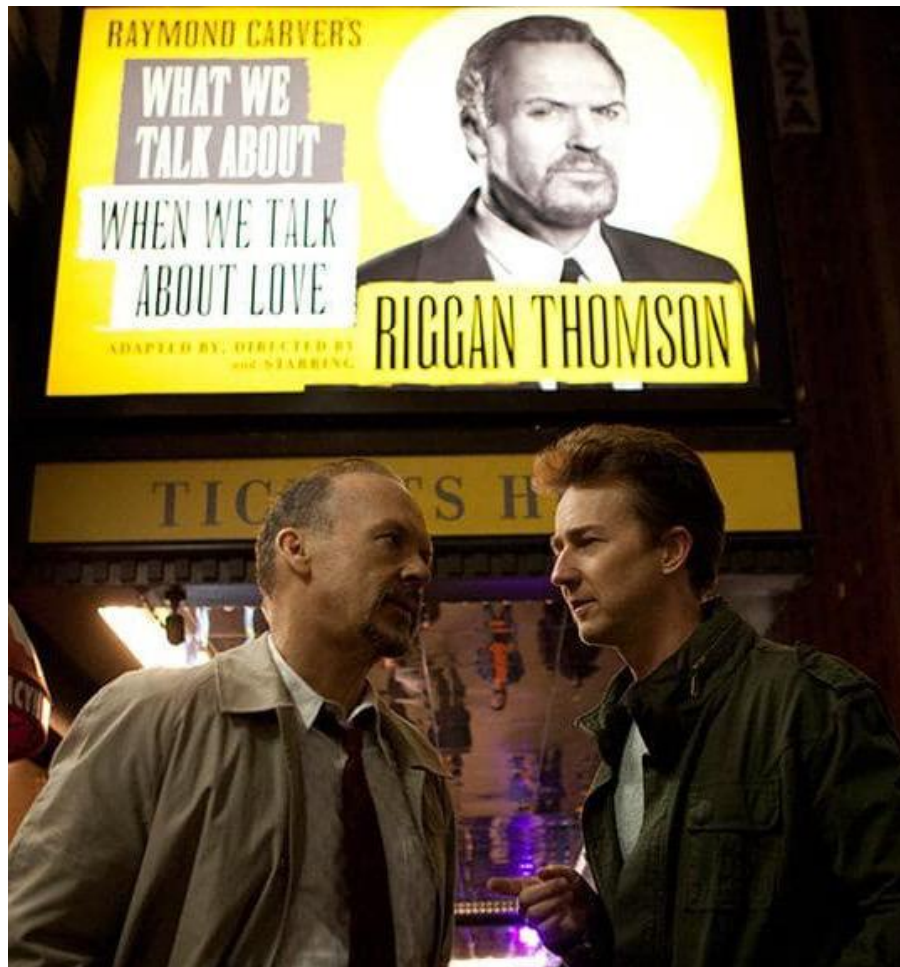
Vulnerability Assessment vs Penetration Test

- Vulnerability Assessment (VA)
 - Vulnerability assessments discover which vulnerabilities are possibly present, but don't try to exploit them
 - Typically automated
- Penetration Test (PT)
 - Penetration tests simulate hacker attempts to get into a system to find exploitable flaws and measure the severity of each
 - Conducted by human beings



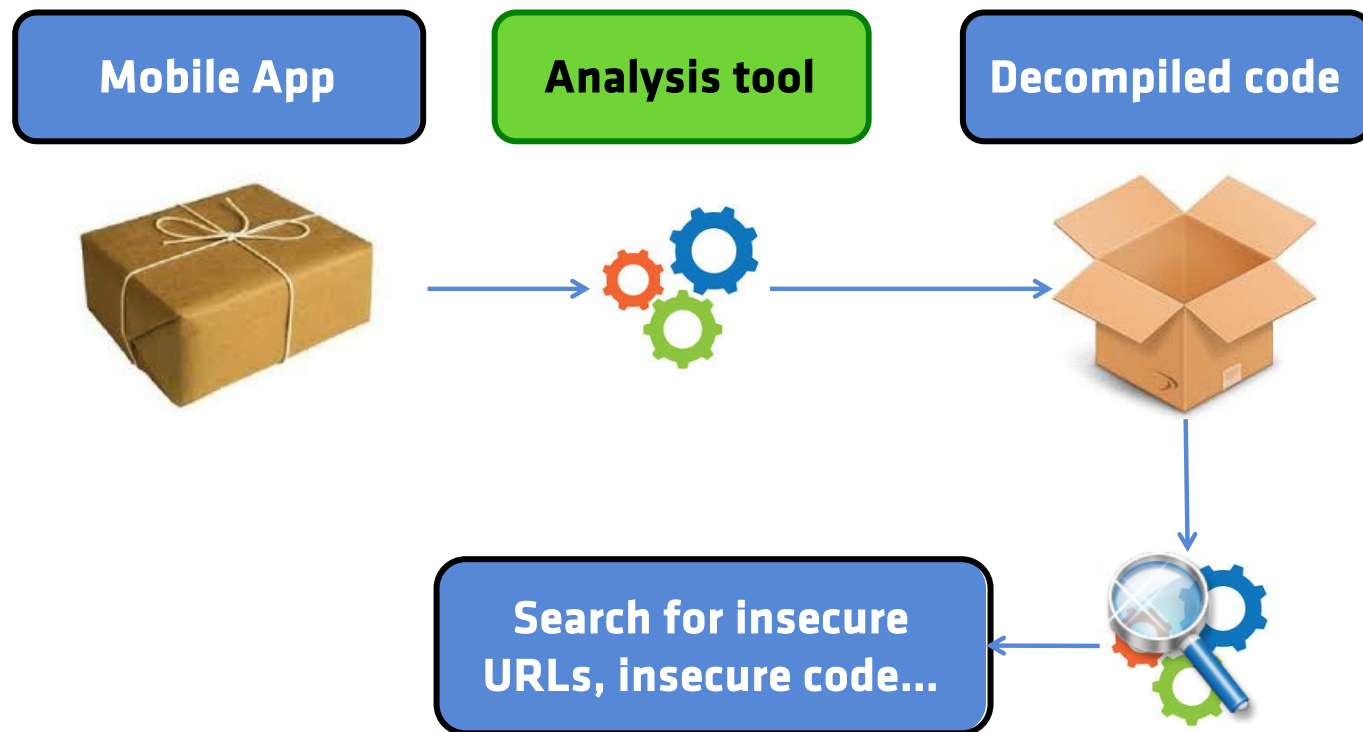
<https://www.alienvault.com/blogs/security-essentials/penetration-testing-vs-vulnerability-scanning-whats-the-difference>

What we talk about when we talk about Mobile App VAPT

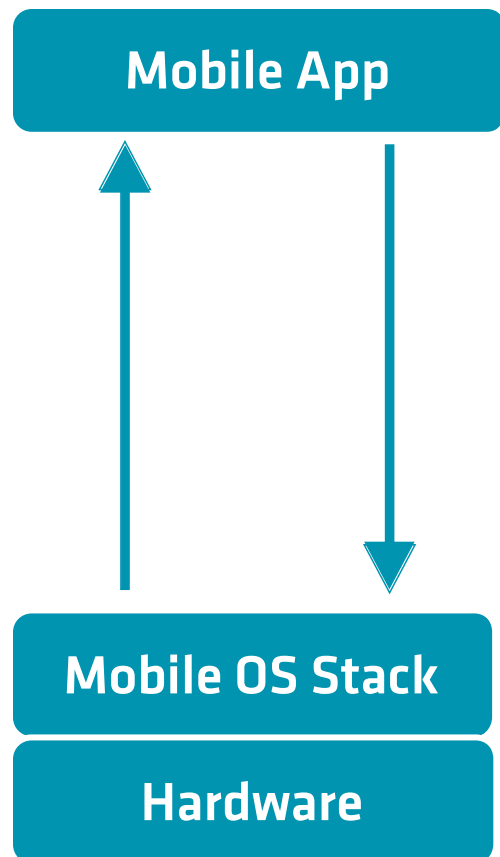


- Static analysis
- Dynamic analysis
- Forensic analysis
- OWASP Mobile Top Ten

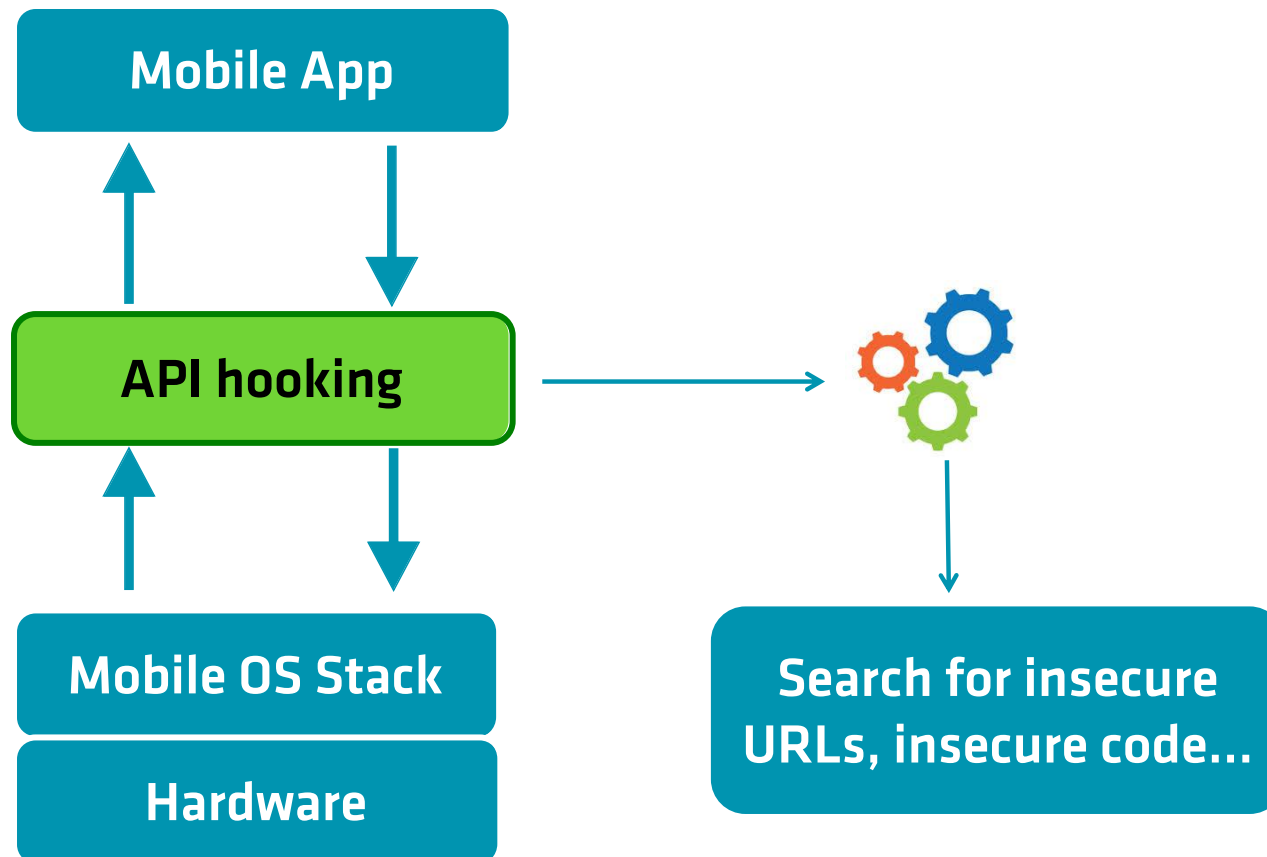
Static Analysis



Dynamic Analysis (app – mobile OS)



Dynamic Analysis (app - mobile OS)



Dynamic Analysis (app – internet)

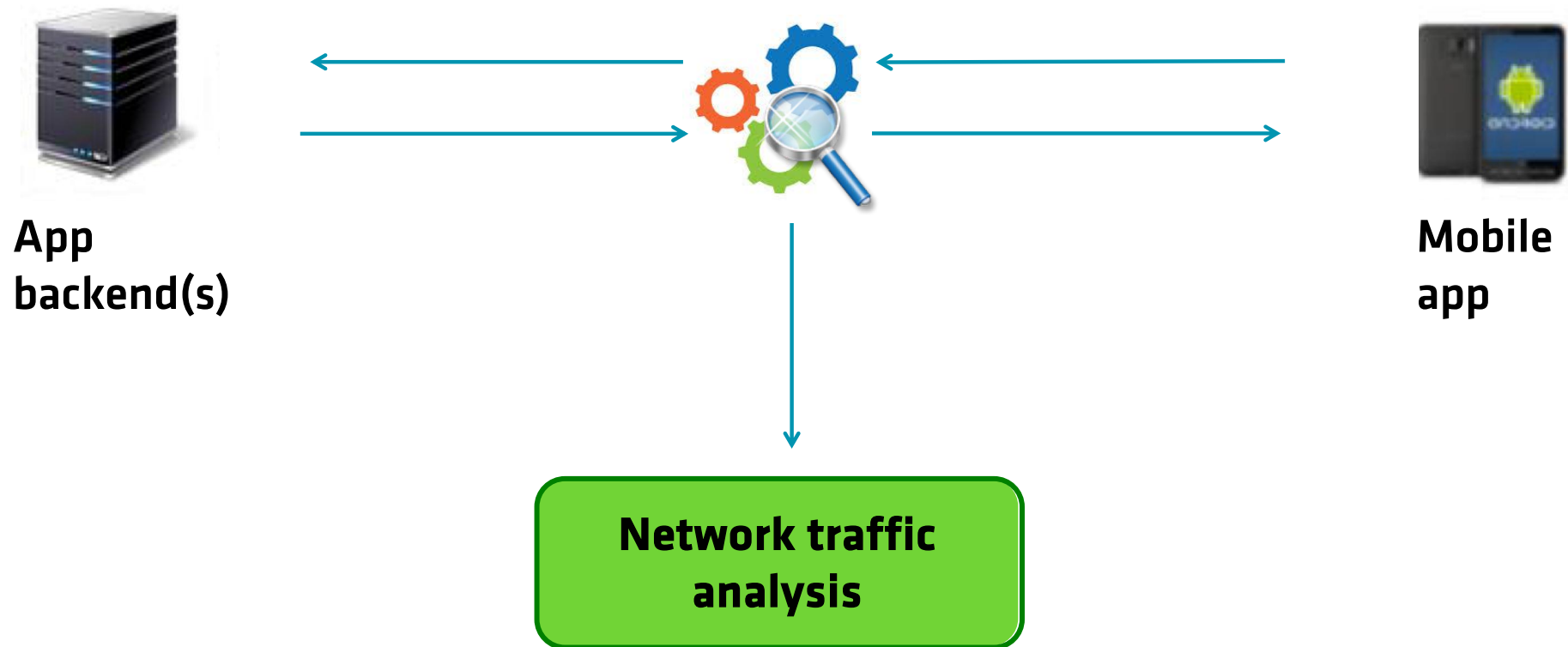


**App
backend(s)**

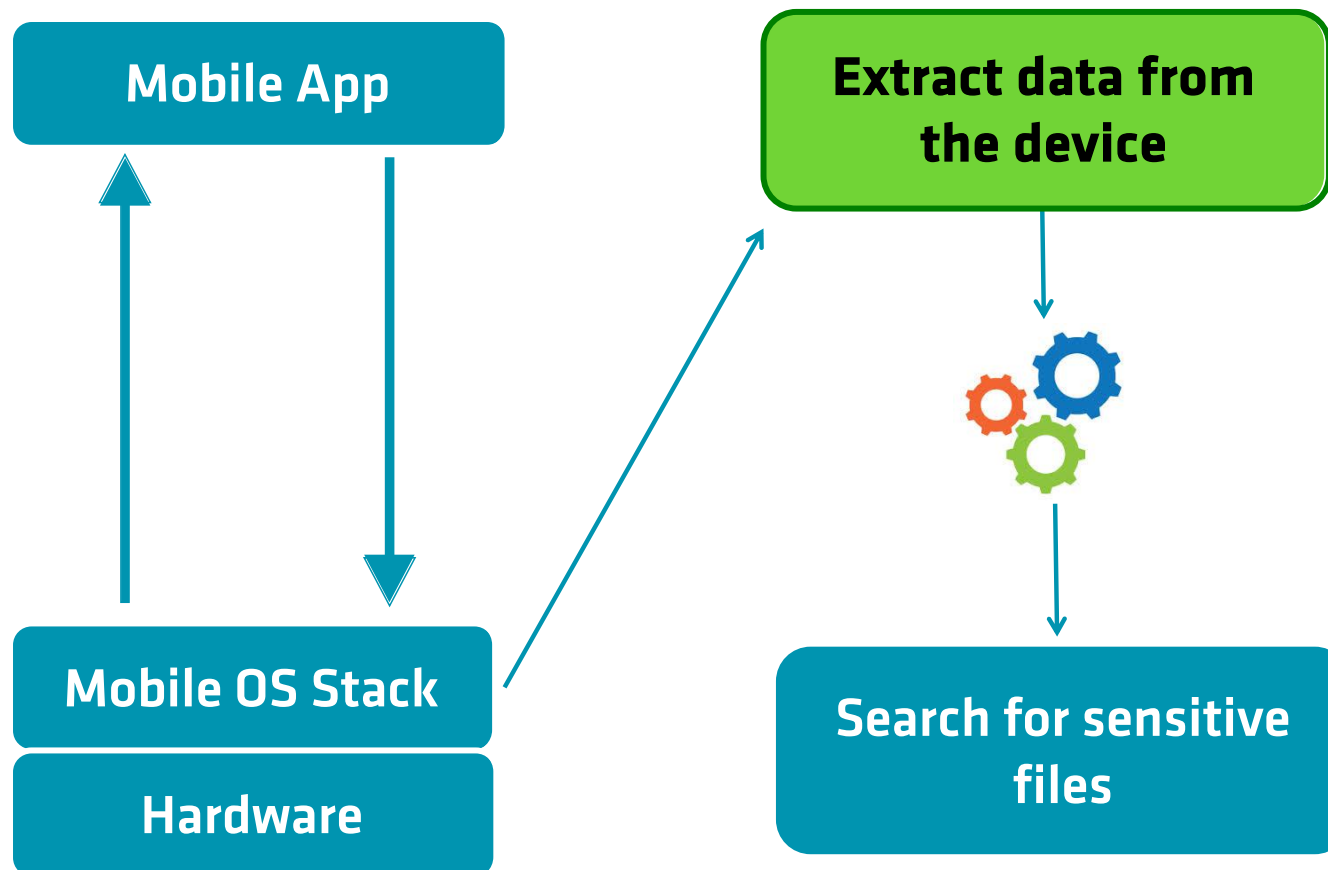


**Mobile
app**

Dynamic Analysis (app - internet)



Forensic Analysis (app – filesystem)



Standard & Benchmark

OWASP Mobile Top 10 (2016)

**M1 – Improper
Platform Usage**

**M2 – Insecure Data
Storage**

**M3 – Insecure
Communication**

**M4 – Insecure
Authentication**

**M5 – Insufficient
Cryptography**

**M6 – Insecure
Authorization**

**M7 – Client Code
Quality**

**M8 – Code
Tampering**

**M9 – Reverse
Engineering**

**M10 – Extraneous
Functionality**



Source: OWASP Mobile
Top 10 2016

M1 – Improper Platform Usage

Misuse of a platform feature or failure to use platform security controls

- Android intents
- Platform permissions
- Misuse of TouchID
- Misuse of the Keychain
- ...

Even server side!



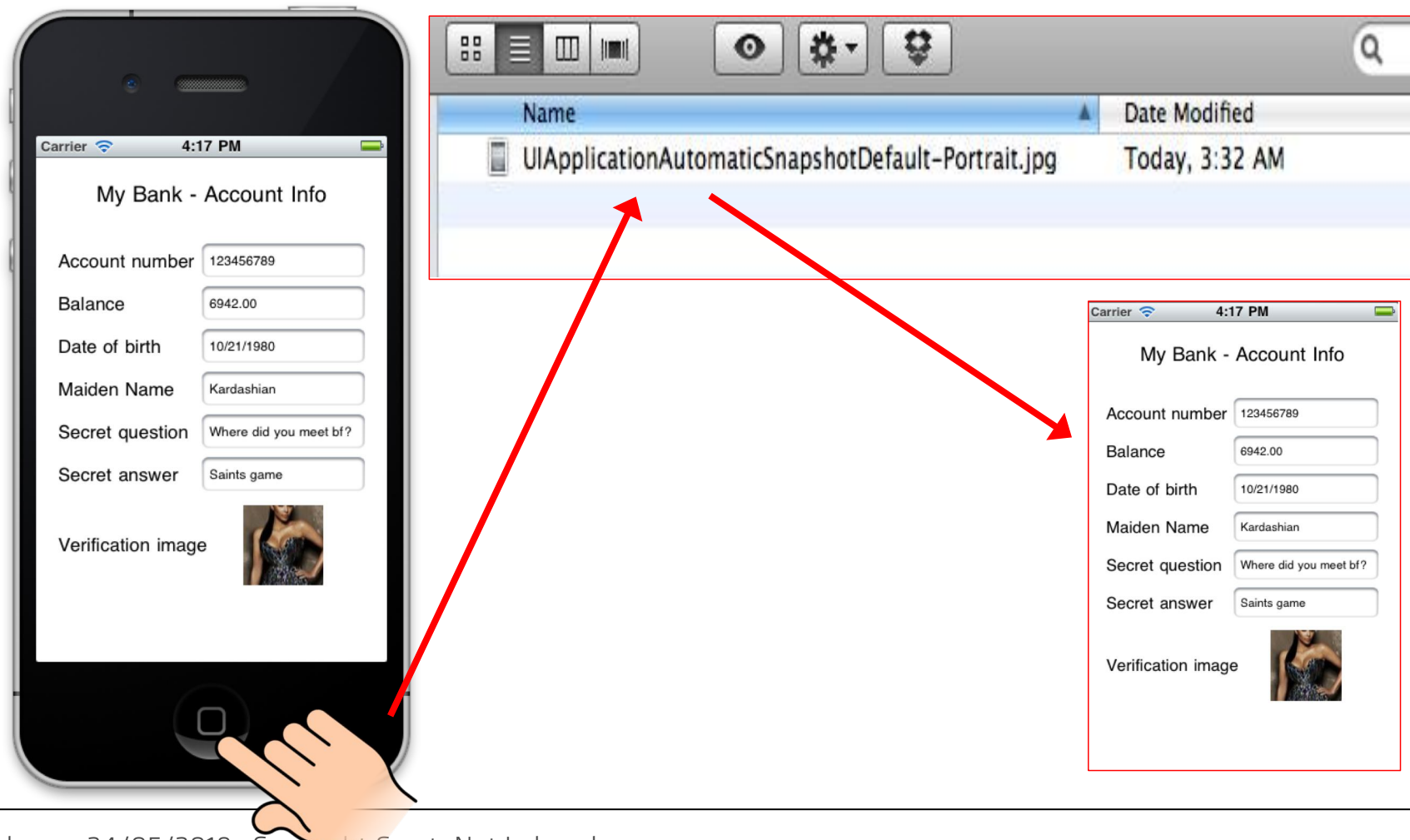
M1 – Improper Platform Usage (unintended data leakage)

Mix of not disabling platform features and programmatic flaws.

Sensitive data ends up in unintended places:

- Logs (system, crash)
- Web caches
- Screenshots (ie: iOS backgrounding)
- Keystroke logging
- Copy/Paste buffer Caching
- Third party servers
- ...

M1 – Improper Platform Usage (unintended data leakage - iOS)



M1 – Improper Platform Usage (unintended data leakage - Log)

```
try {  
    userInfo = client.validateCredentials(userName, password);  
    if (userInfo.get("success").equals("true"))  
        launchHome(v);  
    else {  
        Log.w("Failed login", userName + " " + password);  
    }  
  
} catch (Exception e) {  
    Log.w("Failed login", userName + " " + password);  
}
```


M1 – Improper Platform Usage (TLS)

SSL/TLS:

No SSL, no TLSv1.0, ok TLSv1.1, TLSv1.2

Ciphers:

- No RC2, RC4, Null, Export...
- Possibly no DES
- 128+ bit ciphers
- Forward Secrecy ciphers

Use 2048-bit Private Keys

Signature: no SHA-1, ok SHA-2 family

No Client-Initiated Renegotiation

New Vulnerability Alerts!

M1 – Improper Platform Usage (CVE)

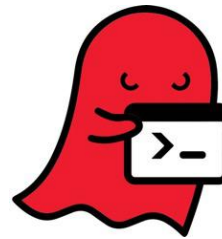
Rely on components (frameworks, libraries, products...) with known vulnerabilities.



Heartbleed
CVE-2014-0160
(OpenSSL)



ShellShock
CVE-2014-6271
(Bash)

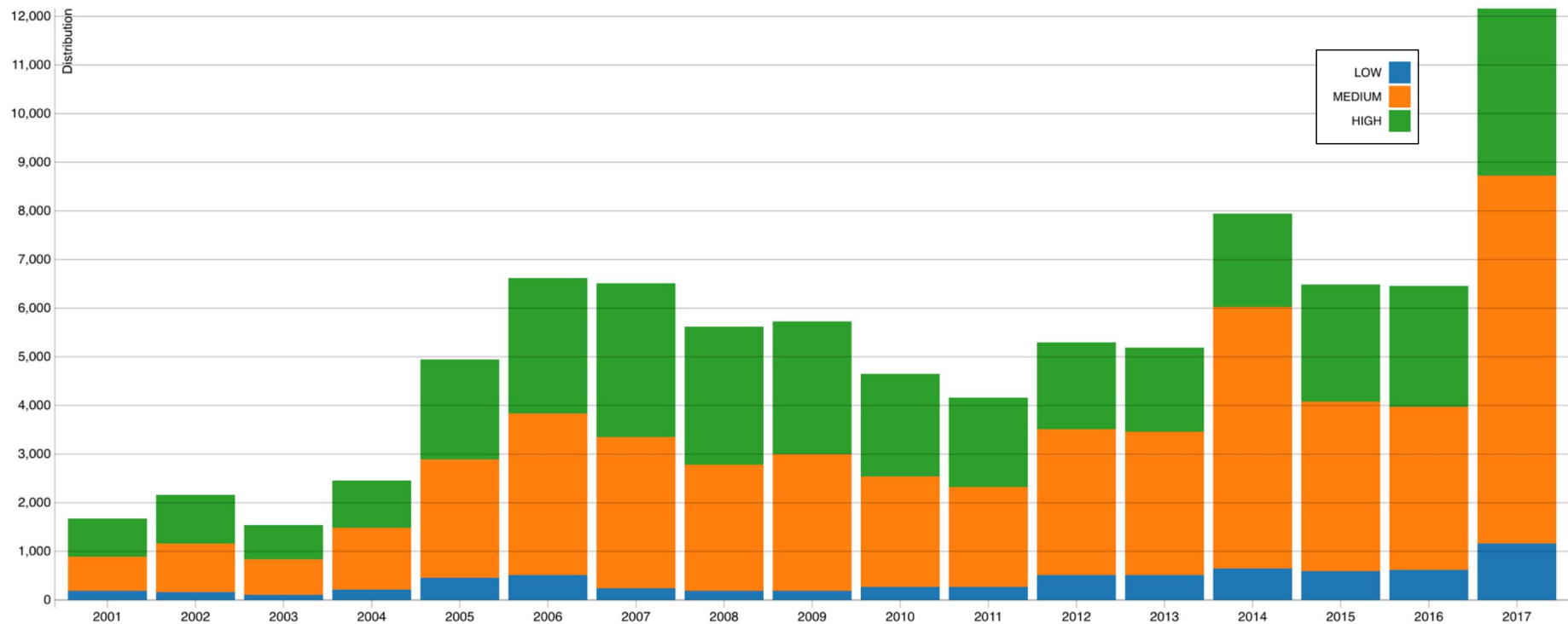


GHOST
CVE-2015-0235
(Linux)



DROWN
CVE-2016-0800
(OpenSSL)

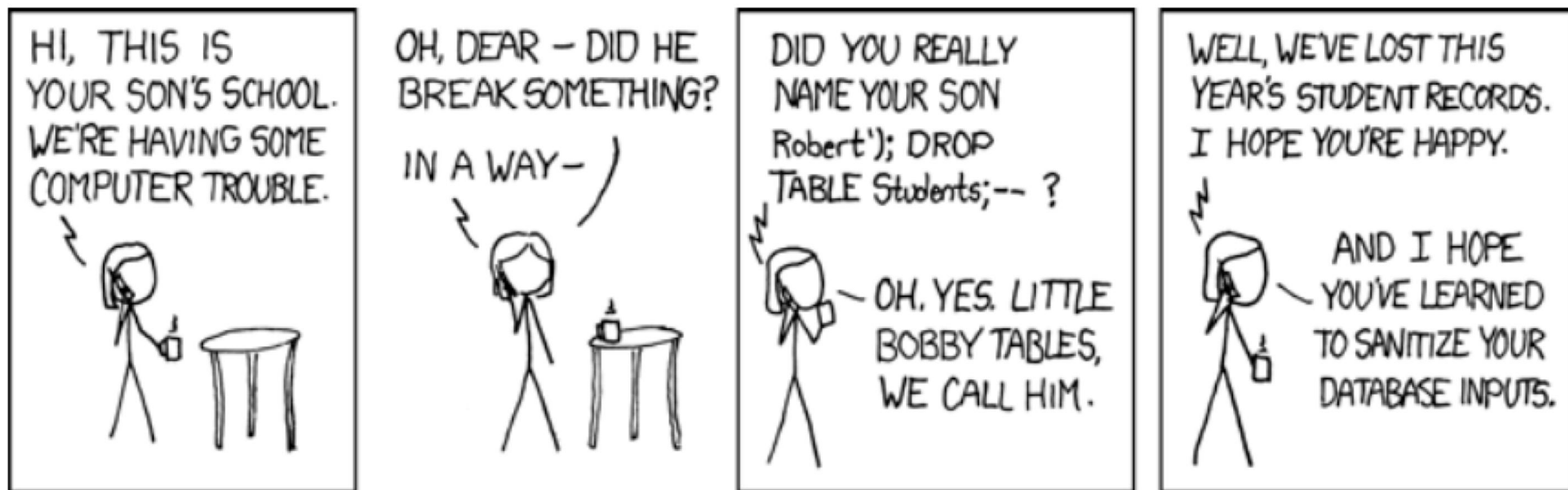
M1 – Improper Platform Usage (CVE)



<https://nvd.nist.gov/vuln-metrics/visualizations/cvss-severity-distribution-over-time>

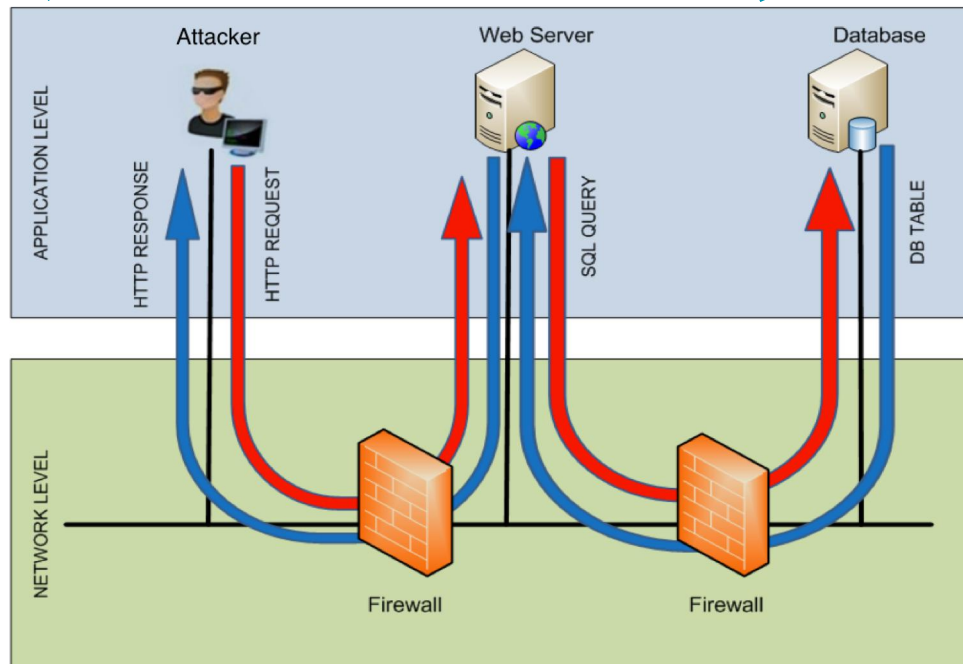
M1 – Improper Platform Usage (server side injection)

Untrusted data is sent to an **interpreter** as part of a command or query. The data can trick the interpreter into **executing unintended commands** or **accessing data** without proper authorization.



<https://xkcd.com/327/>

M1 – Improper Platform Usage (server side SQL injection)



Username <input type="text"/>	String query = "SELECT * FROM accounts WHERE acct=' ' + httpParam + ' '";
Password <input type="text"/>	
<input type="button" value="Submit"/>	↓
	"SELECT * FROM accounts WHERE acct=' ' OR 1=1--"

1. Application presents a form to the attacker
2. Attacker sends an attack in the form data
3. Application forwards attack to the database in a SQL query
4. Database runs query containing attack and sends results back to application
5. Application processes data as normal and sends results to the user

Account Summary

Acct:5424-6066-2134-4334
Acct:4128-7574-3921-0192
Acct:5424-9383-2039-4029
Acct:4128-0004-1234-0293

M1 – Improper Platform Usage

Impact

- Unauthorized access to sensitive data
- Unauthorized actions by malicious users



Recommendations

- Be aware of the technologies you are using
 - On the device
 - On the backend



M2 – Insecure Data Storage

Sensitive data stored on the device in cleartext:

- Usernames
- Authentication tokens
- Passwords
- Cookies
- ...



M2 – Insecure Data Storage (Android)

Writing files with poor permissions:

- Files on /data/data/APP/ with:
MODE_WORLD_READABLE or
MODE_WORLD_WRITEABLE
- Files stored on sdcard
- Allows any app to read or write these files

```
# pwd
/data/data/com.skype.raider
# ls -al files
-rwx----- app_54    app_54      58 2012-08-30 14:44 .flurryagent.-4991a840
drwxrwxrwx app_54    app_54      2012-08-30 14:39 appsec.labs
-rwxrw-rw- app_54    app_54      16 2012-08-30 12:02 csf
-rwxrw-rw- app_54    app_54      0 2012-08-30 12:02 shared.lck
-rwxrw-rw- app_54    app_54    55432 2012-08-30 15:06 shared.xml
-rwxrwxrwx app_54    app_54   2559052 2012-08-30 12:02 skypekit
```

M2 – Insecure Data Storage (Android)

The screenshot displays the file system of an Android application. The directory structure for `com.startsmall.medifile` is shown, including `.DS_Store`, `cache`, `databases` (containing `MediFile.db`), and `lib`. Below the file explorer, three database tables are presented:

Table: MyDoctorVisit

	_id	UserId	DoctorsName	FileName	VisitDetails	IsScheduled	ScheduledDate
1	1	1	Dr. Who		32nd	Yes	2014_11_19

Table: MyPrescription

	_id	UserId	FileName	FileDate
1	1	1	Wrongstor	2014_11_19
2	2	1	Adderall	2014_11_01

Table: MyMedicine

	_id	UserId	FileName	MedicineName	MedicineDosage	Morning
1	1	1		Adderall	100mg	50mg

M2 – Insecure Data Storage

Impact

- Data loss, in the best case, for one user
- In the worst case, for many users



Recommendations

- Store ONLY what is absolutely required
- Never use public storage areas (ie: SD card)
- Never make files publicly readable or writeable
- Use your platforms data security APIs
- Make sure you're calling them appropriately



M3 – Insecure Communication

Complete lack of encryption for transmitted data (HTTP...).

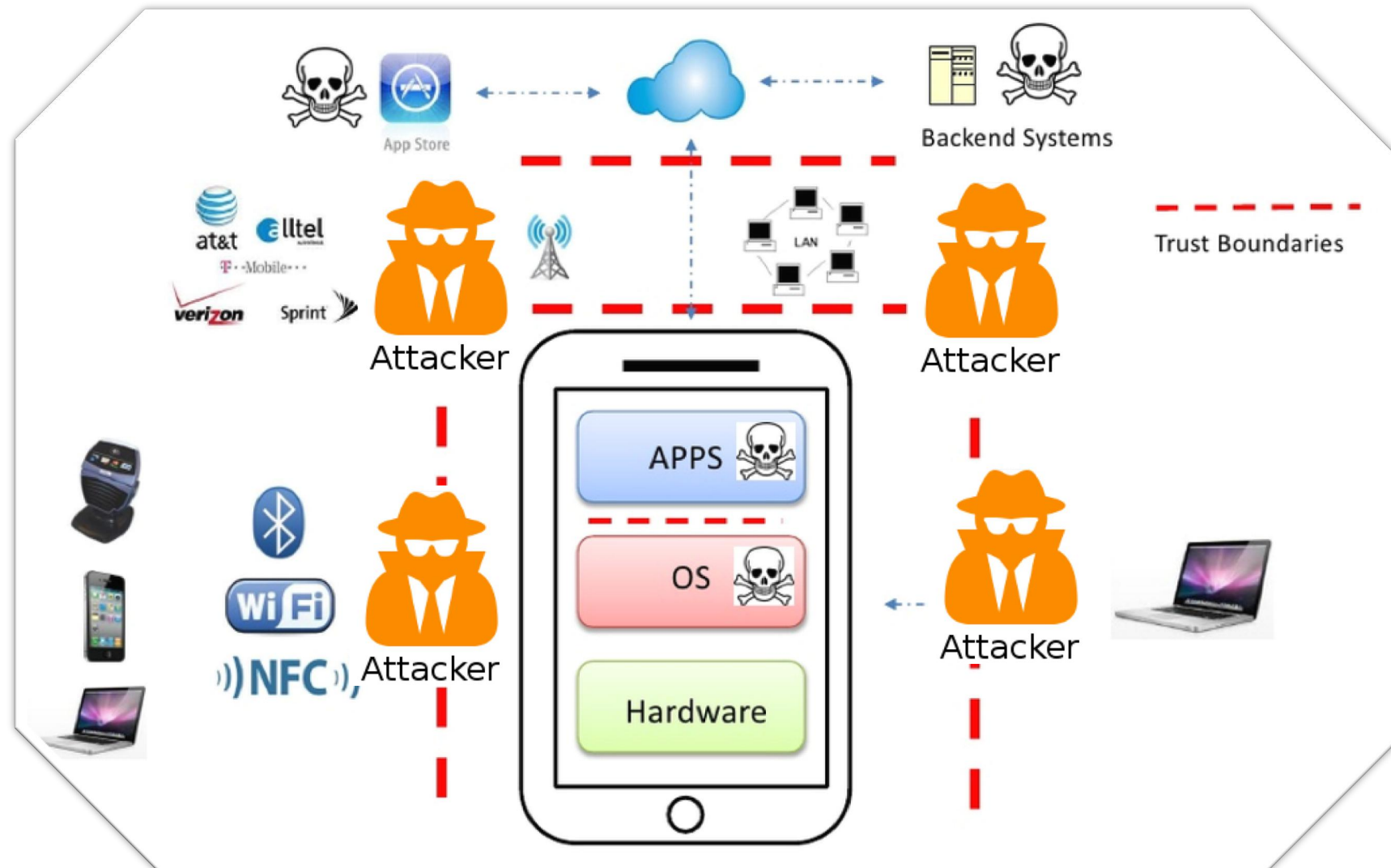


Weakly encrypted data in transit (SSLv3, TLSv1.0...)

Strong encryption, but ignoring security warnings:

- Ignoring certificate validation errors
- Falling back to plain text after failures

M3 – Insecure Communication



M3 – Insecure Communication (Android)

Esempio di classe, ottenuta dalla decompilazione dell'apk, che annulla la verifica dell'hostname:

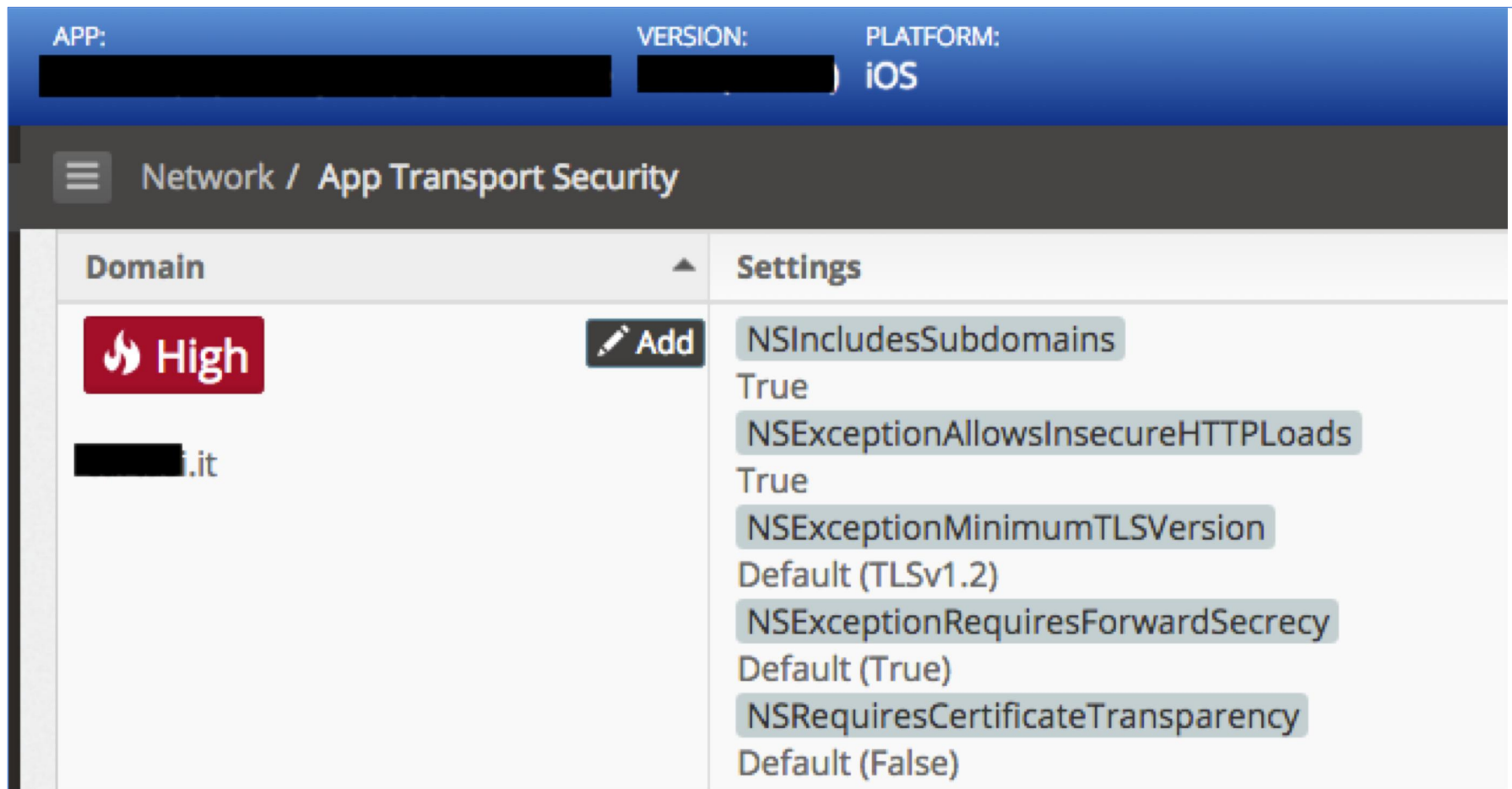
```
package it.██████████.android.api;

import javax.net.ssl.HostnameVerifier;
import javax.net.ssl.SSLSession;

class HttpClient$3
    implements HostnameVerifier
{
    HttpClient$3(HttpClient paramHttpClient)
    {
    }

    public boolean verify(String paramString, SSLSession paramSSLSession)
    {
        return true;
    }
}
```

M3 – Insecure Communication (iOS)



M3 – Insecure Communication

Impact

- Man-in-the-middle attacks:
 - Loss of confidentiality
 - Tampering



Recommendations

- Ensure that all sensitive data leaving the device is encrypted:
 - This includes data over carrier networks, WiFi, and even NFC
- When security exceptions are thrown, it's generally for a reason...DO NOT ignore them!



M3 – Insecure Communication



HTTPS Validation:

- does the subject (CN) of the certificate match the destination selected by the client?
- is the signing CA a trusted CA?
- is the signature correct?
- is the certificate valid in terms of its time of expiry?
- additionally, revocation of a certificate and its corresponding certificate chain should be checked

It's always a good idea not to use libraries written by us:

- Pay attention to the configuration parameters!

M3 – Insecure Communication



Certificate Pinning:

- Hard-code in the client the certificate known to be used by the server
 - A mobile App only needs to connect to a small set of servers
 - Mobile is the ideal platform to implement certificate pinning
- Pin the server's certificate itself
 - Takes the CA system out of the equation
- Pin the CA certificate used to sign the server's certificate
 - Limit trust to certificates signed by one CA or a small set of CAs

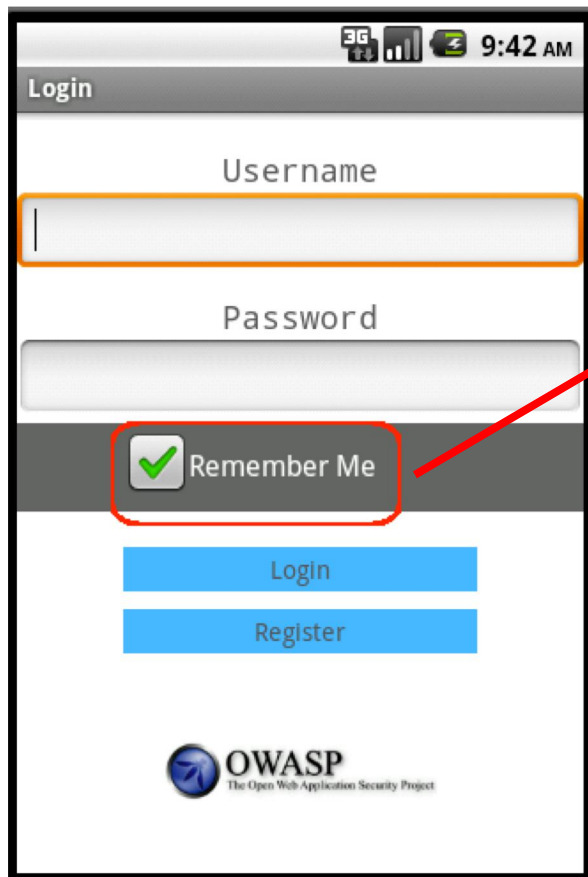
M4 – Insecure Authentication

Authentication issues:

- part architecture
- part mobile
 - Credentials storage
 - Rely on immutable, potentially compromised values (IMEI, IMSI, UUID...)
 - Short passwords due to usability
 - ...



M4 – Insecure Authentication



```
public void saveCredentials(String userName, String password) {  
  
    SharedPreferences credentials = this.getSharedPreferences(  
        "credentials", MODE_WORLD_READABLE); — Very Bad  
    SharedPreferences.Editor editor = credentials.edit();  
    editor.putString("username", userName); — Convenient!  
    editor.putString("password", password);  
    editor.putBoolean("remember", true);  
    editor.commit();  
}
```

M4 – Insecure Authentication

Mobile apps often use session tokens to maintain state over stateless protocols like HTTP or SOAP:

1. Client authenticates with the backend server. **In the request it attaches the Device ID**
- ~~2. Gets a session token in response~~ **The server considers the Device ID as a session token**
- ~~3. Token~~ **Device ID** is added to all requests sent to the server
4. Server can enforce authentication and authorization
- 5. Other apps read the Device ID and impersonate the user**

M4 – Insecure Authentication

Impact

- Unauthorized access



Recommendations

- If possible, perform authentication server-side
- “Remember Me” functionality should never store a user’s password on the device (store a revocable time-limited token)
- If client-side storage of data is required, the data will need to be encrypted using an encryption key that is securely derived from the user’s login credentials



M5 – Insufficient Cryptography

Broken implementations using strong crypto libraries

Custom, easily defeated crypto implementations

Misconceptions:

- Encoding != encryption
- Obfuscation != encryption
- Serialization != encryption



M5 – Insufficient Cryptography (Android)

```
byte[] input = "somedata".getBytes(),
byte[] keyBytes = "SDF$%$1KSF23".getBytes();
SecretKeySpec key = new SecretKeySpec(keyBytes, "DES");
Cipher cipher = Cipher.getInstance("DES/ECB/PKCS7Padding", "BC");
cipher.init(Cipher.ENCRYPT_MODE, key);
byte[] cipherText = new byte[cipher.getOutputSize(input.length)];
int ctLength = cipher.update(input, 0, input.length, cipherText, 0);
ctLength += cipher.doFinal(cipherText, ctLength);
```

Hard
coded
key

Bad
algorithm

Bad
crypto
mode

M5 – Insufficient Cryptography

Impact

- Loss of confidentiality of data
- Circumvent business logic



Recommendations

- Storing the key with the encrypted data negates everything
- Use your platforms APIs or well-known third party libraries
 - Make sure you're calling them appropriately



M6 – Insecure Authorization

Authorization issues:

- part architecture
- part mobile
 - Rely on immutable, potentially compromised values (IMEI, IMSI, UUID...)
 - Rely on secret parameters
 - ...



M6 – Insecure Authorization

Controlling vehicle features of Nissan LEAFs across the globe via vulnerable APIs

← → ↺ 🏠 www.troyhunt.com/2016/02/controlling-vehicle-features-of-nissan.html

But again, it's passive data – is the climate control on or off and as a result, what should the buttons say. But then he tried turning it on and observed this request:

```
GET https://[redacted].com/orchestration_1111/gdc/ACRemoteRequest.php?RegionCode=NE&lg=no-  
NO&DCMID=&VIN=SNJFAAZE0U60XXXXX&tz=Europe/Paris
```

That request returned this response:

```
{  
  status: 200,  
  message: "success",  
  userId: "[redacted]",  
  vin: "SNJFAAZE0U60[redacted]",  
  resultKey: "[redacted]"  
}
```

M6 – Insecure Authorization

Impact

- Privilege escalation
- Unauthorized access



Recommendations

- Always rely only on server-side authorization
- Ensure to integrate authorization in every functionality of the app



M7 – Client Code Quality



Untrusted data is sent to an **interpreter** on the **device** as part of a command or query.

The data can trick the interpreter into executing unintended commands or accessing data without proper authorization.

Input can come from:

- Another app (iOS: url scheme, Android: intent/content provider)
- Shared file (ie: sdcard) manipulated by another app
- Server side response
- 3rd party web site

M7 – Client Code Quality

SQL Injection

embedding untrusted input into raw SQL statements:

```
String query = "select * from table where  
columnName='\"+external_input+\"'";  
db.rawQuery( query, null );
```

Command Injection

embedding untrusted input into OS command execution:

```
Process process = Runtime.getRuntime().exec("top -n " + external_input  
);
```

M7 – Client Code Quality

Skype iOS URL Scheme Handling Issue

1. Attacker crafts malicious website containing this code:
`<iframe src="skype://14085555555?call"></iframe>`
2. User visits the website with Safari
3. Safari throws no warning, and yanks the user into Skype
4. Skype initiates phone call without asking for user consent



<http://software-security.sans.org/blog/2010/11/08/insecure-handling-url-schemes-apples-ios/>

M7 – Client Code Quality

Impact

- Device compromise
- Privilege escalation
- Jailbreak!



Recommendations

- Restrict access to IPC communications to a white-list of trusted apps
- Sanitize or escape untrusted data before rendering or executing it
- Use prepared statements for database calls (no concatenation!)



M8 – Code Tampering

The app executable lacks protection against:

- Malicious Root Apps
- Unauthorized Code Modification



M8 – Code Tampering

Impact

- User data exposure
- Code modifications (piracy, malware...)



Recommendations

- Jailbreak/root detection
- Anti-tampering
- Anti-debugging
- Stack Smashing Protection (iOS)
- Position Independent Executable (iOS)
- Automatic Reference Counting (iOS)



M9 – Reverse Engineering

The app executable lacks protection against Analysis and Reverse Engineering



M9 – Reverse Engineering

Impact

- User data exposure
- Code exposure (secret algorithms, secret keys...)



Recommendations

- Obfuscation: renaming, string encryption, control flow obfuscation
- Anti-debugging



M10 – Extraneous Functionality



App contains functionalities and data that are not intended to be released into a production environment:

- Hidden backdoors
- Hardcoded passwords
- Private IPs
- Private API keys
- Test code
- ...

M10 – Extraneous Functionality

Impact

- Privilege escalation
- Sensitive data exposure

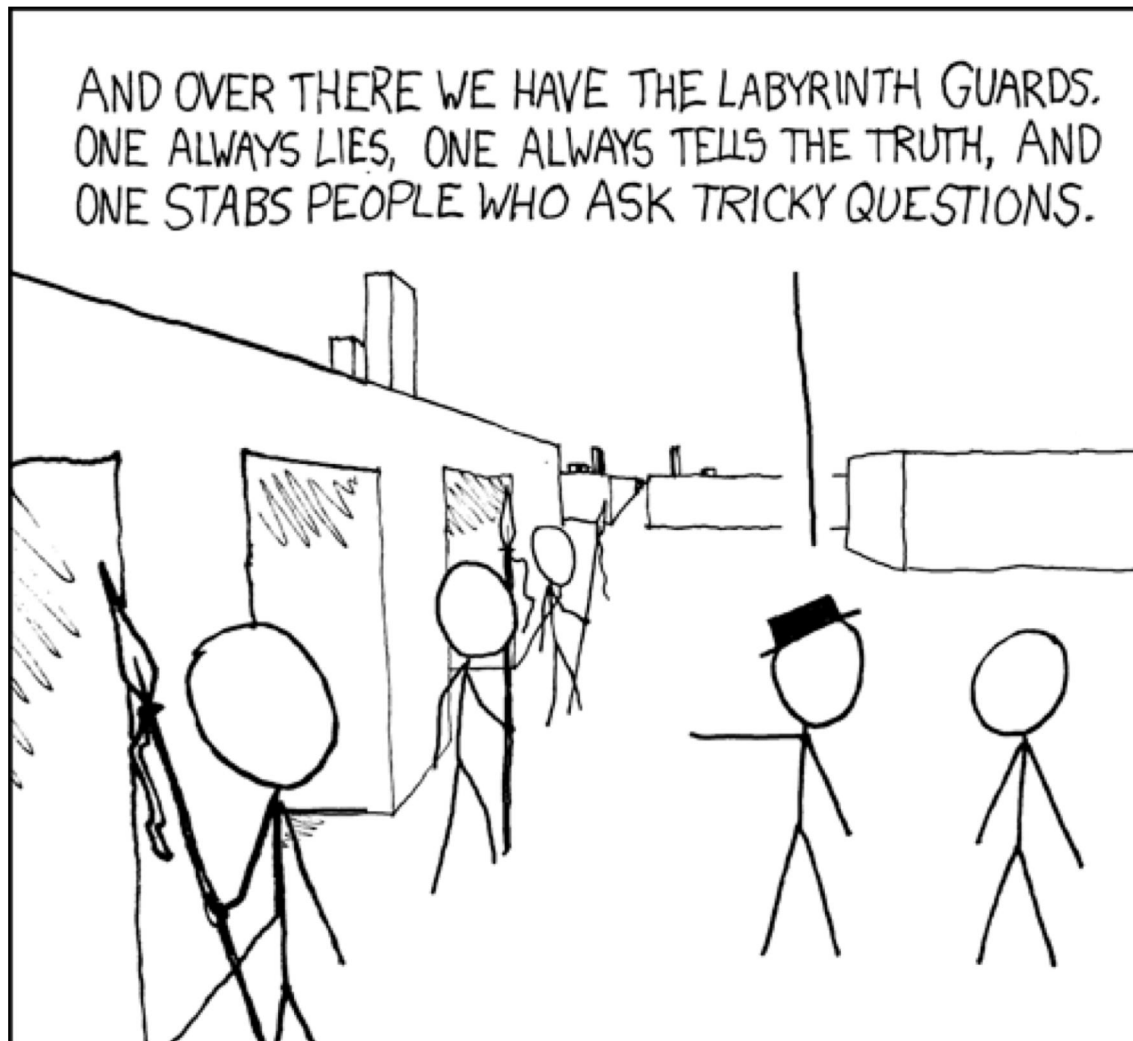


Recommendations

- Examine the app's configuration settings to discover any hidden switches
- Verify that all test code is not included in the final production build of the app



luca.capacci@cryptonetlabs.it



<https://xkcd.com/246/>