Module 18
Hacking Mobile Platform

Ansh Bhawnani

Mobile Attack Vectors

Module 18

1. Mobile Threat Report

Module 18

Mobile Attack Vectors

- The mobile app Fortnite with its 200 million players and 60 million downloads is a fertile ground for fake apps disguised as versions of the game.
- In 2018, TimpDoor, while not new, became the leading mobile backdoor family by more than double and a solid example of how tried and true phishing over SMS is still popular among cyber criminals to deceive users into installing malware.
- Banking trojans on mobile devices has continued to rise, particularly homed in on account holders of both large and regional banks.
- Cyber criminals are looking to find ways to add value to their digital wallets without the cost of doing their own mining.
- Spyware attacks spike on mobile are an attractive target for nation-state threat actors to gain intelligence and track victims.

2. Terminology

Module 18

Mobile Attack Vectors

- **Stock ROM**: It is the default ROM (operating system) of an android device supplied by the manufacturer©
- **CyanogenMod**: It is a modified device ROM without the restrictions imposed by device's original ROM©
- **Bricking the Mobile Device**: Altering the device OSes using ***rooting*** or ***jailbreaking*** in a way that causes the mobile device to become unusable or inoperable©
- **Bring Your Own Device** (BYOD): Bring your own device (BYOD) is a business policy that allows employees to bring their personal mobile devices to their workplace.

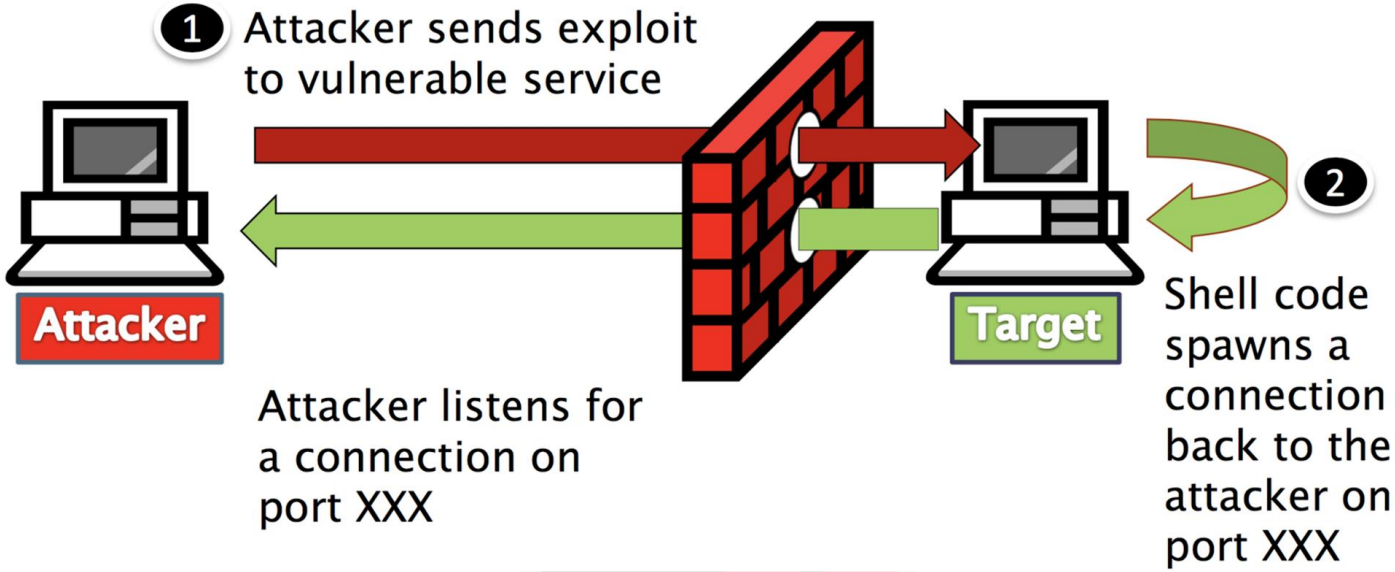
Mobile Attack Vectors

- **Metasploit** is one of the most powerful tools used for penetration testing. Most of its resources can be found at – **www.metasploit.com**.
- It comes in two versions: commercial and free edition.
- The **hardware requirements** to install Metasploit are –
  - 2 GHz + processor
  - 1 GB RAM available
  - 1 GB + available disk space

Mobile Attack Vectors

- The recommended **Software** OS versions for Metasploit are –
  - Kali Linux 2.0 or Upper Versions
  - Red Hat Enterprise Linux Server 7.1+
  - Ubuntu Linux 14.04 LTS
  - Windows Server 2012 R2
  - Windows 10

Mobile Attack Vectors



Mobile Attack Vectors

- **Armitage** is a complement tool for Metasploit. It visualizes targets, recommends exploits, and exposes the advanced post-exploitation features. Armitage is incorporated with Kali distribution.
- Armitage is very user friendly. Its GUI has three distinct areas:
  - The area **Targets** lists all the machines that you have discovered and those you are working with. The hacked targets have red color with a thunderstorm on it.
  - The area **Console** provides a view for the folders. Just by clicking on it, you can directly navigate to the folders without using any Metasploit commands.
  - The area **Modules** is the section that lists the module of vulnerabilities.

Mobile Attack Vectors

- **Exploit:** After vulnerability scanning and vulnerability validation, we have to run and test some scripts, in order to gain access to a machine and do what we are planning to do.
  - **Active Exploits**: They will exploit a specific host, run until completion, and then exit.
    - Brute-force modules will exit when a shell opens from the victim.
    - Module execution stops if an error is encountered.
    - You can force an active module to the background by passing '-j' to the exploit command.

Mobile Attack Vectors

- **Passive Exploits:** Passive exploits wait for incoming hosts and exploit them as they connect.
  - Passive exploits almost always focus on clients such as web browsers, FTP clients, etc.
  - They can also be used in conjunction with email exploits, waiting for connections.
  - Passive exploits report shells as they happen can be enumerated by passing '-l' to the sessions command. Passing '-i' will interact with a shell.

Mobile Attack Vectors

- Payload, in simple terms, are simple scripts that the hackers utilize to interact with a hacked system. Using payloads, they can transfer data to a victim system. Metasploit payloads can be of three types –
  - **Singles** – Singles are very small and designed to create some kind of communication, then move to the next stage. For example, just creating a user.
  - **Staged** – It is a payload that an attacker can use to upload a bigger file onto a victim system.
  - **Stages** – Stages are payload components that are downloaded by Stagers modules. The various payload stages provide advanced features with no size limits such as Meterpreter and VNC Injection.

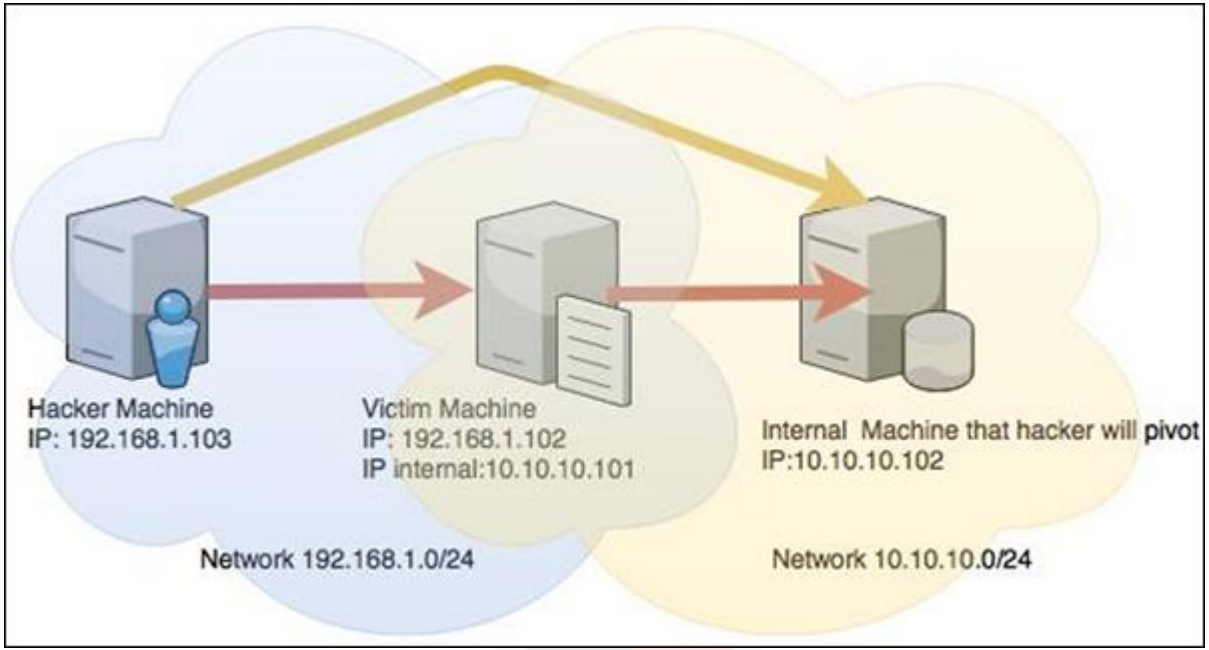
Mobile Attack Vectors

- Metasploit currently has over 547 payloads. Some of them are:
  - **Command shell** enables users to run collection scripts or run arbitrary commands against the host.
  - **Meterpreter** enables users to control the screen of a device using VNC and to browse, upload and download files.
  - **Dynamic payloads** enables users to evade anti-virus defense by generating unique payloads.
  - **Static payloads** enables static IP address/port forwarding for communication between the host and the client system.

Mobile Attack Vectors

- **Pivoting** is a technique that Metasploit uses to route the traffic from a hacked computer toward other networks that are not accessible by a hacker machine.
- A network with the range 192.168.1.0/24 where the hacker machine has access, and
- Another network with the range 10.10.10.0/24. It is an internal network and the hacker doesn't have access to it.
- The hacker will try to hack the second network this machine that has access in both networks to exploit and hack other internal machines.
- Hacker will first break into the first network and then use it as a staging point to exploit and hack the internal machines of the second network. This process is known as **pivoting.**

Mobile Attack Vectors



Mobile Attack Vectors

- **Backdoor**: After going through all the hard work of exploiting a system, it's often a good idea to leave yourself an easier way back into it for later use. This way, if the service you initially exploited is down or patched, you can still gain access to the system.

3. Mobile Platform Vulnerabilities and Risks

Module 18

Mobile Attack Vectors

- Weak Server Side Controls
- Lack of Binary Protections
- Insecure Data Storage
- Insufficient Transport Layer Protection
- Unintended Data Leakage
- Poor Authorization and Authentication
- Broken Cryptography
- Client Side Injection
- Security Decisions via Untrusted Inputs
- Improper Session Handling

Hacking Android OS

Module 18

Introduction

Module 18

Mobile Attack Vectors

- Android is an open source and Linux-based **Operating System** for mobile devices such as smartphones and tablet computers. Android was developed by the ***Open Handset Alliance***, led by Google, and other companies.
- The first beta version of the Android Software Development Kit (SDK) was released by Google in 2007 where as the first commercial version, Android 1.0, was released in September 2008.
- The source code for Android is available under free and open source software licenses. Google publishes most of the code under the ***Apache License*** version 2.0 and the rest, Linux kernel changes, under the ***GNU General Public License*** version 2.
- Android applications are usually developed in the Java language using the Android ***Software Development Kit***.

Mobile Attack Vectors

Android 1.6 Donut

Android 2.0 Eclair

Android 2.2 Froyo

Android 2.3 Gingerbead

Android 3.0 Honeycomb

Android 4.0 Ice Cream Sandwich

Android 4.1 Jelly Bean

Android 4.4 KitKat

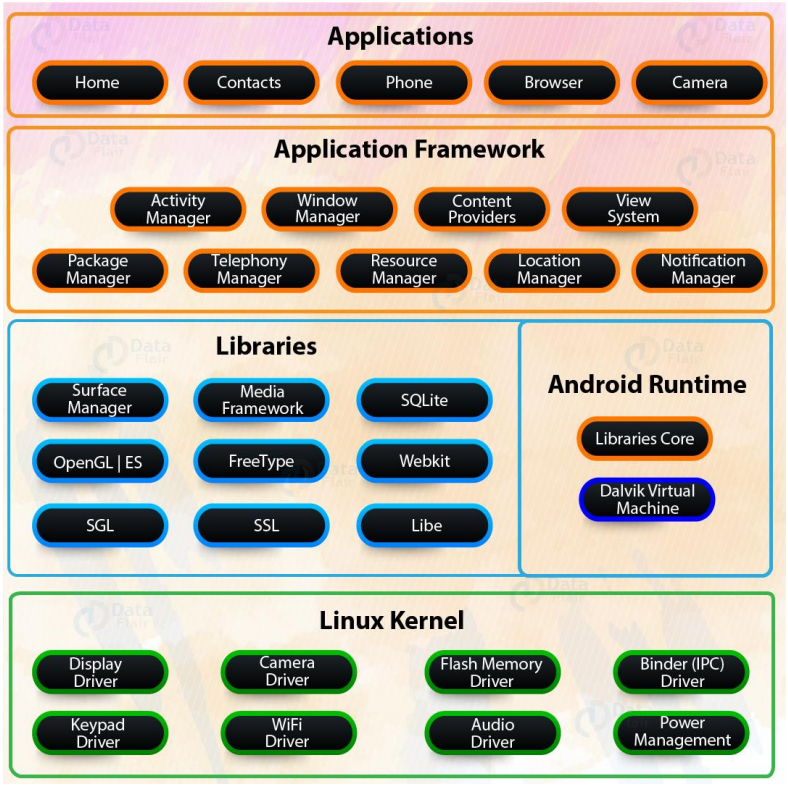
Android 5.0 Lolipop

Android 6.0 Marshmallow

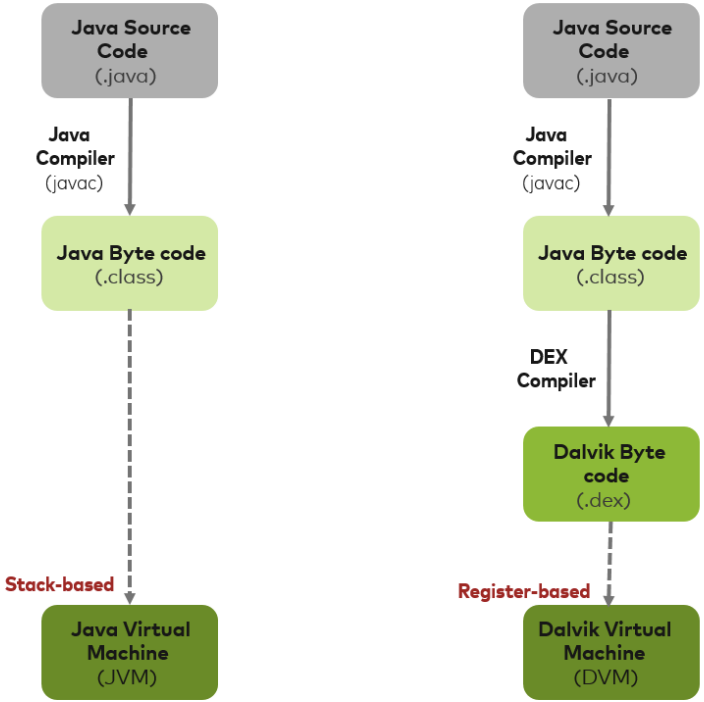
1. Android OS Architecture

Module 18

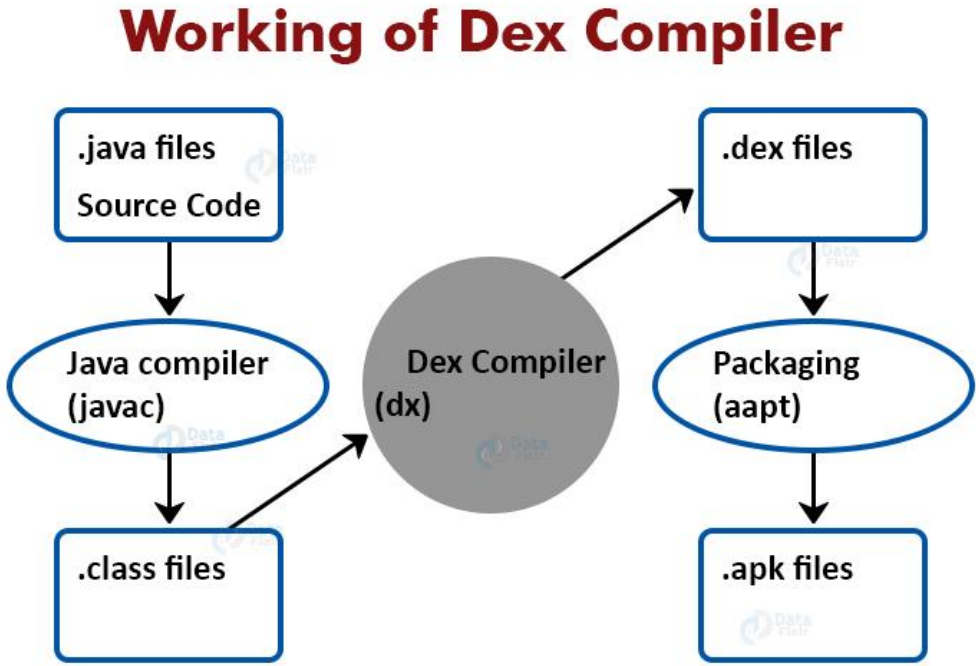
Hacking Android OS



Hacking Android OS



Hacking Android OS



Hacking Android OS

Linux kernel

- At the bottom of the layers is Linux - Linux 3.6 with approximately 115 patches. This provides a level of abstraction between the device hardware and it contains all the essential hardware drivers like camera, keypad, display etc.
- Also, the kernel handles all the things that Linux is really good at such as networking and a vast array of device drivers, which take the pain out of interfacing to peripheral hardware.

Hacking Android OS

Libraries

- On top of Linux kernel there is a set of libraries including open-source Web browser engine WebKit, well known library libc, SQLite database which is a useful repository for storage and sharing of application data, libraries to play and record audio and video, SSL libraries responsible for Internet security etc.

Android Libraries

- This category encompasses those Java-based libraries that are specific to Android development. Examples of libraries in this category include:
  - **android.app**
  - **android.content**
  - **android.opengl**
  - **android.os**
  - **android.widget**
  - **android.webkit**

Hacking Android OS

Android Runtime

- This is the third section of the architecture and available on the second layer from the bottom. This section provides a key component called **Dalvik Virtual Machine** which is a kind of Java Virtual Machine specially designed and optimized for Android.
- The Dalvik VM makes use of Linux core features like memory management and multi-threading, which is intrinsic in the Java language. The Dalvik VM enables every Android application to run in its own process, with its own instance of the Dalvik virtual machine.

Hacking Android OS

Application Framework

- The Application Framework layer provides many higher-level services to applications in the form of Java classes. Application developers are allowed to make use of these services in their applications. The Android framework includes the following key services –
  - **Activity Manager**
  - **Content Providers**
  - **Resource Manager**
  - **Notifications Manager**
  - **View System**

Hacking Android OS

Applications

- You will find all the Android application at the top layer. You will write your application to be installed on this layer only. Examples of such applications are Contacts Books, Browser, Games etc.

2. Android Rooting

Module 18

Hacking Android OS

- **Rooting** is about obtaining root access to the underlying Linux system beneath Android and thus gaining absolute control over the software that is running on the device.
- Things that require root access on a typical Linux system —
  - mounting and unmounting file systems,
  - starting your favorite SSH or HTTP or DHCP or DNS or proxy servers,
  - killing system processes, chroot-ing,
  - Being able to run arbitrary commands as the root allows you to do absolutely ***anything*** on a Linux / Android system

Hacking Android OS

- **Stock OEM** Android builds typically do not allow users to execute arbitrary code as root.
- The **bootloader**, the first piece of code executed when your device is powered on, is responsible for loading the Android OS and the recovery system and flashing a new ROM.
- Many Android devices have locked bootloaders that you would have to hack around in order to make them do anything other than boot the stock ROM.
- **System recovery** is the second piece of low-level code on board any Android device. It is separate from the Android userland and is typically located on its own partition; it is usually booted by the bootloader when you press a certain combination of hardware keys.

Hacking Android OS

- However, since recovery is stored in a partition just like **/system**, **/data** and **/cache,** you can replace it with a custom recovery if you have root access in Linux / Android.
- **ADB** allows a PC or a Mac to connect to an Android device and perform certain operations. One such operation is to launch a simple shell on the device, using the command **adb shell**.
- If **ro.secure=0,** an ADB shell will run commands as the root user on the device. But if **ro.secure=1**, an ADB shell will run commands as an unprivileged user on the device.
- The value of this property is set at boot time from the **default.prop** file in the root directory.

Hacking Android OS

- The contents of the root directory are essentially ***copied*** from a partition in the internal storage on boot, but you cannot write to the partition if you are not already root. So the only way you could change it is by gaining root access in the first place.
- On an Android system, all Android applications that you can see or interact with directly are running as **_un_privileged** users in sandboxes.
- On Linux, privilege escalation is usually accomplished via the **su** and **sudo** programs; they are often the only programs in the system that are able to execute the system call **setuid(0)** that changes the current program from running as an unprivileged user to running as root.

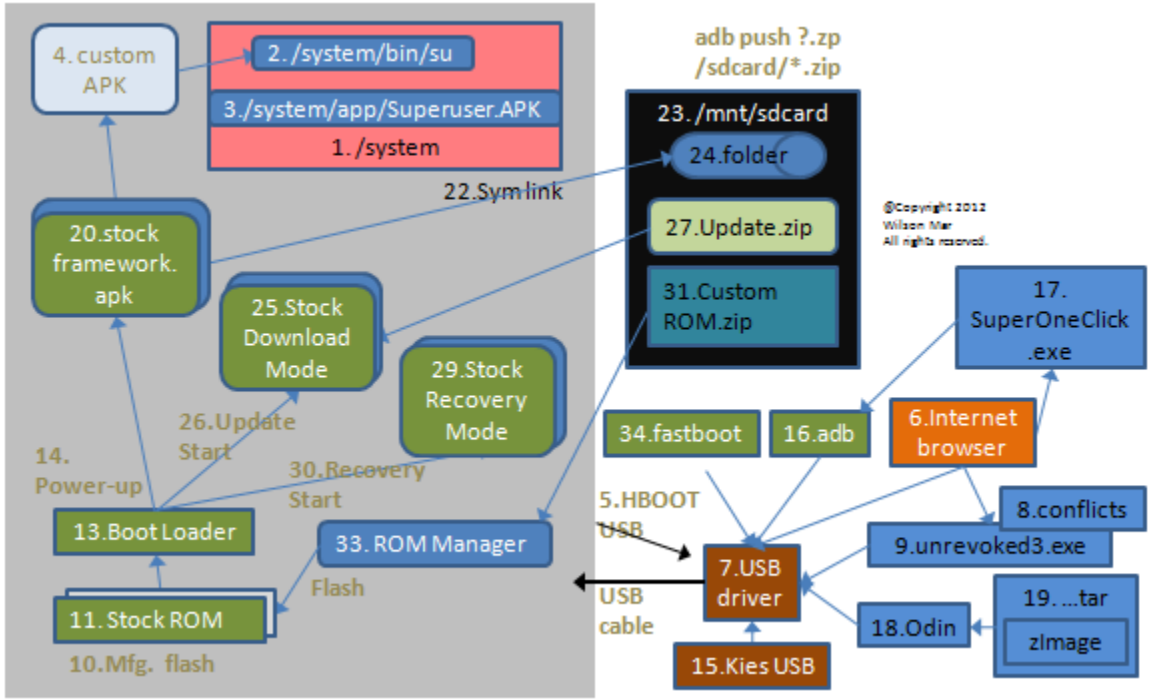
Hacking Android OS

- Unsurprisingly, stock OEM ROMs never come with these su. You cannot just download it or copy it over either; it needs to have its SUID bit set, which indicates to the system that the programs this allowed to escalate its runtime privileges to root.
- To summarize, what this means is that any program that you can interact with on Android (and hence running in unprivileged mode) is unable to either 1) gain privileged access and execute in privileged mode, or 2) start another program that executes in privileged mode.

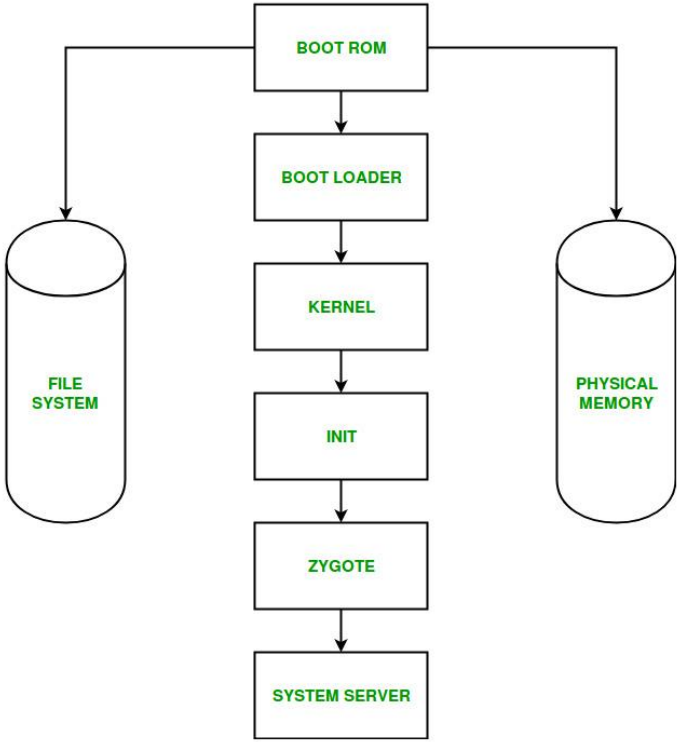
Hacking Android OS

- Typing **ps** on an Android shell (either via ADB or a terminal emulator on the device) will give you programs started by the init process, the first process started by the kernel (the kernel spawns init in a particular fashion, and init then goes on and spawns all other processes) which has to run as root because it needs to start other privileged system processes.
- If you can hack / trick one of these system processes running in privileged mode to execute your arbitrary code, you have just gained privileged access to the system.
- This how all one-click-root methods work, including **z4root**, **gingerbreak**, and so on.
- "Arbitrary code" is most certainly a piece of code that mounts /system in read-write mode and installs a copy of su permanently on the system

Hacking Android OS



Hacking Android OS



Hacking Android OS

Advantages:

- Support for themes, allowing everything to be visually changed even while the device is booting,
- Full control of the kernel, which, for example, allows overclocking and underclocking the CPU and GPU.
- Full application control, including the ability to backup, restore, or batch edit applications, or to remove bloatware
- Custom automated system-level processes
- Ability to install a custom firmware or ROM or software (such as Xposed, BusyBox, etc.)

Hacking Android OS

- **Disadvantages**:
  - Voids the phone warranty
  - Risk of "bricking" a phone.
  - Breaks the phone contract.
  - Poor performance.
  - Viruses.

Android Penetration Testing

Module 18

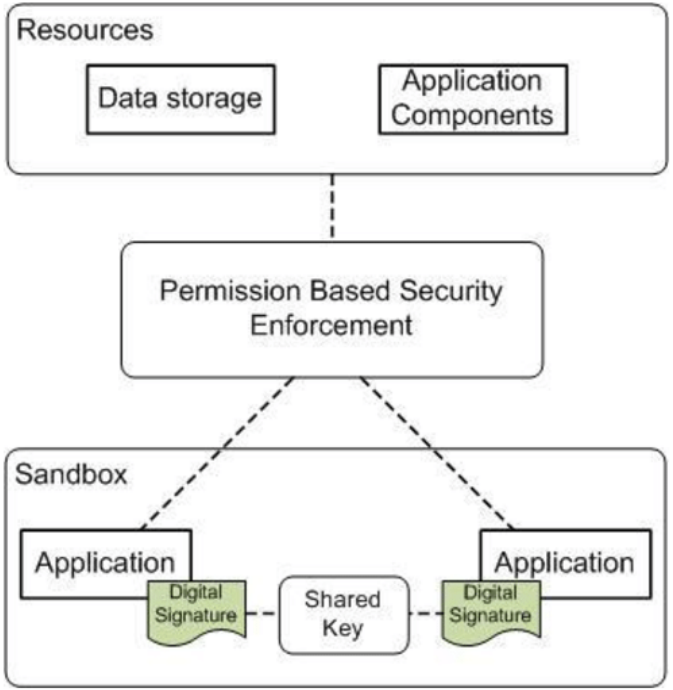
Android Security Architecture

Module 18

Android Penetration Testing

- Android provides these **key security features**:
  - Robust security at the OS level through the Linux kernel
  - Mandatory app sandbox for all apps
  - Secure interprocess communication
  - App ***signing***
  - App-defined and user-granted ***permissions***

Android Penetration Testing



Android Penetration Testing

Permission control

Privilege control
Process control

Android Penetration Testing

- Every application is given a separate user ID and process ID
- User of that application is the owner of that PID

```
C:\Windows\system32>adb shell ps
USER      PID   PPID  VSIZE  RSS     WCHAN    PC         NAME
root      1     0     760    392   ffffffff 00000000 S /init
root      2     0     0      0     ffffffff 00000000 S kthreadd
root      3     2     0      0     ffffffff 00000000 S ksoftirqd/0
root      5     2     0      0     ffffffff 00000000 S kworker/0:0H
root      7     2     0      0     ffffffff 00000000 S migration/0
root      8     2     0      0     ffffffff 00000000 S rcu_preempt
root      9     2     0      0     ffffffff 00000000 S rcu_bh
root      10    2     0      0     ffffffff 00000000 S rcu_sched
root      11    2     0      0     ffffffff 00000000 R migration/1
root      12    2     0      0     ffffffff 00000000 R ksoftirqd/1
root      14    2     0      0     ffffffff 00000000 S kworker/1:0H
root      15    2     0      0     ffffffff 00000000 R migration/2
root      16    2     0      0     ffffffff 00000000 R ksoftirqd/2
root      18    2     0      0     ffffffff 00000000 S kworker/2:0H
root      19    2     0      0     ffffffff 00000000 R migration/3
root      20    2     0      0     ffffffff 00000000 R ksoftirqd/3
root      22    2     0      0     ffffffff 00000000 S kworker/3:0H
root      23    2     0      0     ffffffff 00000000 S khelper
root      24    2     0      0     ffffffff 00000000 S suspend_sys_syn
root      25    2     0      0     ffffffff 00000000 S suspend
root      26    2     0      0     ffffffff 00000000 S writeback
root      27    2     0      0     ffffffff 00000000 S bioset
root      28    2     0      0     ffffffff 00000000 S kblockd
root      29    2     0      0     ffffffff 00000000 S khubd
root      48    2     0      0     ffffffff 00000000 S irq/322-charger
```

Android Penetration Testing

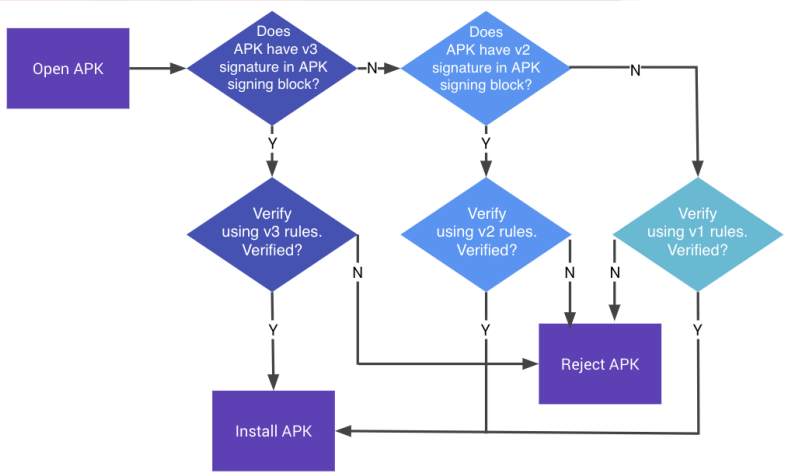
- A very important and compulsory file present in every Android App is "*AndroidManifest.xml*".
  - It primarily describes the application's activities, services and broadcast receivers.
  - Some declarations in it let the Android OS know what components the app has and when there is a need to launch them.
  - It declares which permissions the application needs for accessing the protected parts.
  - It also declares the permissions that other apps require to have in order to interact with the application's components.

Android Penetration Testing

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="payspace.ssidit.pp.ua.payspacemagazine">

    <uses-permission android:title="android.permission.INTERNET" />
    <uses-permission android:title="android.permission.ACCESS_NETWORK_STATE" />

    <application
        android:allowBackup="true"
        android:icon="@mipmap/ic_launcher"
        android:label="@string/app_name"
        android:theme="@android:style/Theme.Holo.Light">
        <activity
            android:label="@string/app_name"
            android:title=".MainActivity">
            <intent-filter>
                <action android:title="android.intent.action.MAIN" />

                <category android:title="android.intent.category.LAUNCHER" />
            </intent-filter>
        </activity>
        <activity
            android:label="@string/title_activity_settings"
            android:title=".SettingsActivity"></activity>
    </application>

</manifest>
```

Android Penetration Testing

- **App Signing**: The developer is identified by this signature and the private key is also held by him only. The purpose of this certificate is to distinguish the authors and allow the system to grant or deny signature-level permissions.



Android Application Components

Module 18

Android Penetration Testing

- **Basic** Components
  - Activity
  - Intent
  - Service
  - Content Provider
- **Additional** Components
  - Fragments
  - Views
  - Layouts
  - Resources

Setting up your Lab

Module 18

1. Attacking Machine

Module 18

Android Penetration Testing

Santoku OS

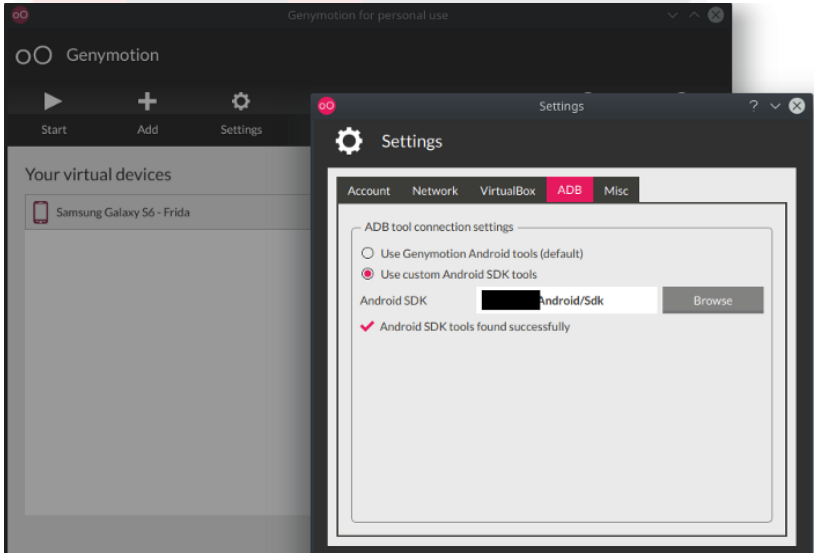
- Santoku is dedicated to mobile forensics, analysis, and security, and packaged in an easy to use, Open Source platform.

2. Client Machine (Android Device)

Android Penetration Testing

Genymotion Android Emulator

- If you don't have an Android device, probably you need an emulator. I prefer using Genymotion For Fun but you can use other applications as well.



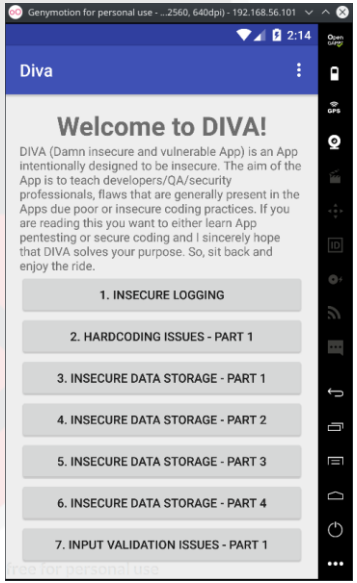
3. Testing Application

Module 18

Android Penetration Testing

DIVA

- There are lots of APK files for penetration testing in Android OS but mostly we will use DIVA application.



4. Communication Toolkit

Module 18

Android Penetration Testing

ADB

- Android Debug Bridge (adb) is a versatile command-line tool that lets you communicate with a device.
- The adb command facilitates a variety of device actions, such as installing and debugging apps, and it provides access to a Unix shell that you can use to run a variety of commands on a device.
- It is a client-server program that includes three components:

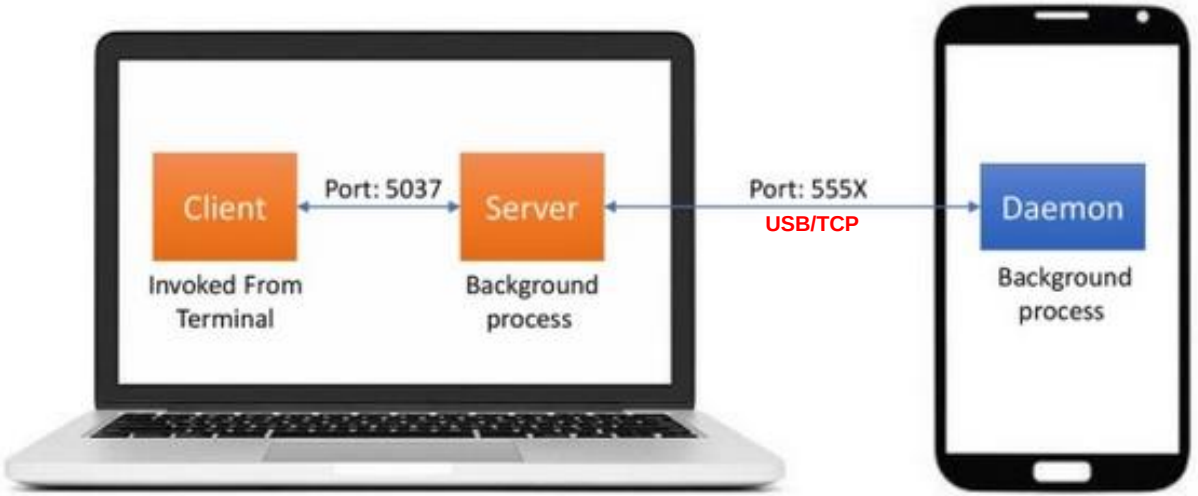
Android Penetration Testing

- **A client**, which sends commands. The client runs on your development machine. You can invoke a client from a command-line terminal by issuing an ***adb*** command.
- **A daemon (adbd)**, which runs commands on a device. The daemon runs as a background process on each device.
- **A server**, which manages communication between the client and the daemon. The server runs as a background process on your development machine.

How adb works

- When you start an adb client, the client first checks whether there is an adb server process already running. If there isn't, it starts the server process. When the server starts, it binds to local TCP port 5037 and listens for commands sent from adb clients—all adb clients use port 5037 to communicate with the adb server.
- The server then sets up connections to all running devices. It locates emulators by scanning odd-numbered ports in the range 5555 to 5585, the range used by the first 16 emulators. Where the server finds an adb daemon (adbd), it sets up a connection to that port.

Android Penetration Testing



5. Reverse Engineering tools

Module 18

Android Penetration Testing

Apktool

- A tool for reverse engineering 3rd party, closed, binary Android apps. It can decode resources to nearly original form and rebuild them after making some modifications.
- It also makes working with an app easier because of the project like file structure and automation of some repetitive tasks like building apk, etc.
- Decompiles to ***Smali***, can't get Java source code from apk.

Android Penetration Testing

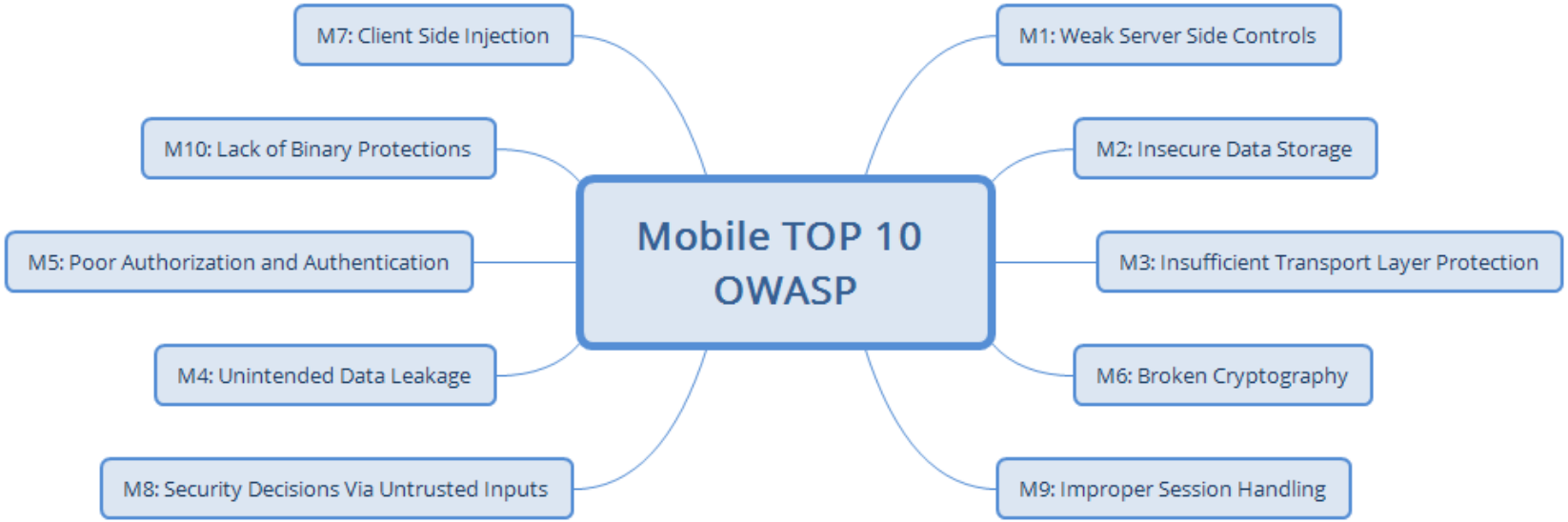
JaDX

- It is a tool that produces Java source code from Android DEX and APK files.
- Allows you to see the app structure after decompiling.
- It's licensed under ***Apache 2.0***.
- If the app uses some non-ASCII characters the decompilation will fail.

6. Mobile OWASP Top 10

Module 18

Android Penetration Testing



Android Penetration Testing

Insecure Logging

- Logging is a method that developers use for tracing the code and watching warnings or errors.
- These logs are stored in a ***central repository*** for all the apps to have access to.
- Logging any sensitive data can cause this issue.

Android Penetration Testing

Hardcoding issues

- Developers may leave plaintext strings in the app source code containing raw data such as ***API keys***, ***access tokens***, ***passwords***, etc.
- We can recover this sensitive data by simple reverse engineering the source code.

Android Penetration Testing

Insecure Data Storage

- Developers store sensitive info in plaintext on local storage without encryption.
- Ways to store data locally:
  - *Shared preferences*
  - *Databases*
  - *Temp files*
  - *External Storage*

Input Validation Issues (SQL Injection)

- It occurs when there is improper or no input sanitization by the application against SQL queries.
- Attacker can run SQLi commands to manipulate ***SQLite*** databases.

Android Penetration Testing

Abusing WebView

- Android WebView is used to display web page in android
- In the android, every message between applications is as a **URL**.
- Attacker can supply URLs with ***file://*** protocol to access any file on the android device.

Android Penetration Testing

Access Control Issues

- Developers often fail to check access control in every activity.
- Android ***ActivityManager*** allows us to open any activity with an ***Intent Filter***.
- We can bypass authorization by directly opening the privileged activity.

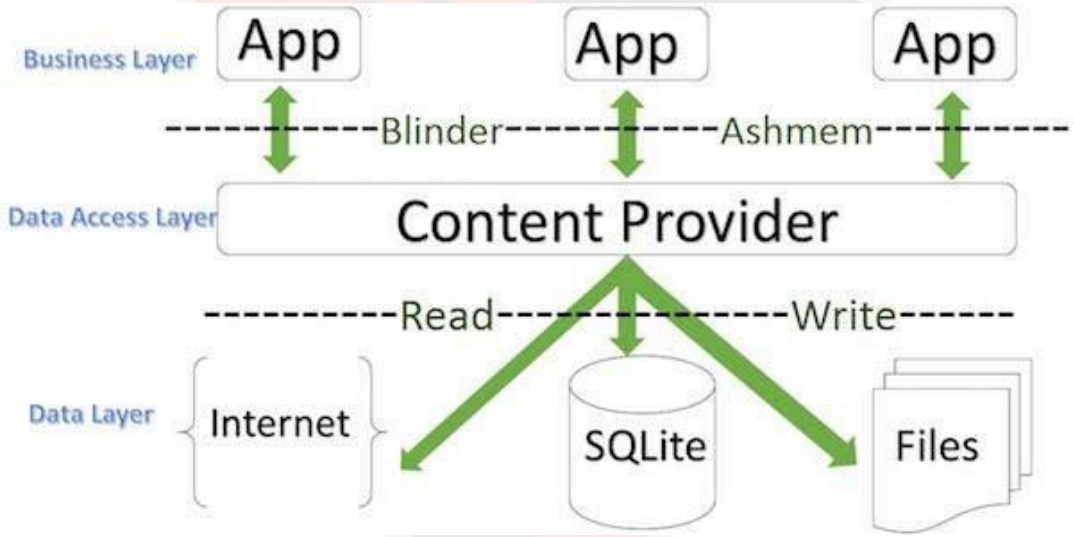
Android Penetration Testing

Leaking Content Provider

- A content provider is required if you need to share data between multiple applications.
- A special form of URI which starts with "**content://**" is assigned to each content provider
- Any app which knows this URI can ***insert***, ***update***, ***delete***, and ***query*** data from the database of the provider app.
- If proper security controls are not enforced in the app, that leads to leakage of information.

Android Penetration Testing

Leaking Content Provider



Hacking iOS

Module 18

1. Jailbreaking iOS

Module 18

Hacking iOS

- **Jailbreak** actually means to allow third-party applications to be installed into your iDevice. Contrary to popular beliefs, it's actually fully legal to run third-party applications on your device since James H. Billington's DMCA revision. Having this in mind the only thing that prevents us from having an easy jailbreak is Apple.
- Jailbreak itself is getting control over the root and media partition of your iDevice; where all the iOS's files are stored at. To do so ***/private/etc/fstab*** must be patched.
- **fstab** is the switch room of your iDevice, controlling the permission of the root and media partition. The default is 'read-only', allowing eyes and no hands. To be able to modify the root and media partition we must set the fstab to 'read-write', allowing eyes and hands

Hacking iOS

- The **main** problem is not getting the files in, but getting them through various checkpoints. These checkpoints were put by Apple to verify if the file is indeed legit, or a third-party. Every file is signed by a key, and without it, the file will be put aside and be unusable.

Hacking iOS

- When an **iDevice boots up** it goes trough a "chain of trust". This chain is a series of signature checks that makes sure everything being ran is Apple approved. It goes on the following (specific) order:
  - **Runs Bootrom**: Also called "SecureROM" by Apple, it is the first significant code that runs on an iDevice.
  - **Runs Bootloader**: Generally, it is responsible for loading the main firmware.
  - **Loads Kernel**: Bridge between the iOS and the actual data processing done at the hardware level.
  - **Loads iOS**: The final step to the chain, iOS starts and we get our nice "Slide to Unlock" view.

Hacking iOS

- What is the **roadblock** in a jailbreak?
  - What prevents an easy jailbreak is the signature checks. While the kernel is loading there are thousands of checks being done to make sure everything being loaded is Apple approved.
  - To be more specific, there are many checks through out the boot which look for one thing, a signature, a key. If the key is correct we get a green light, if it is wrong, depending where the check was at or what file it was, it will either crash the iDevice causing a loop, or simply ignore it and does not execute that specific file at all.

Hacking iOS

- Jailbreaking **objective** is to either patch the checks or bypass them. This brings us to two broad categories of exploits:
  - **bootrom exploit:** Exploit done during the bootrom. It **must** be patched by new hardware. Since it's before almost any checkpoint, the malicious code is injected before everything, thus allowing a passageway to be created to bypass all checks or simply disable them.
  - **userland exploit:** Exploit done during or after the kernel has loaded and can easily be patched by Apple with a software update. Since it's after all the checks, it injects the malicious code directly into the openings back into the kernel. These openings are not so easy to find, and once found can be patched.

Hacking iOS

- How did some of the released jailbreak actually worked?
  - **Limera1n** (exploit, not tool): Bootrom exploit first used by Geohot. Due to it being a bootrom it can't be patched by Apple with a software update, which means it is still usable today in all A4 devices. Yes... including iOS 6.
  - **JailbreakMe**: Userland exploit that used a malformed CFF vulnerability. CFF stands for Compact Font Format and it's used to store fonts. Starting with PDF version 1.2 it could be embedded directly into the .pdf file, but it had it's malfunctions.

2. Jailbreaking vs. Android Rooting

Hacking iOS

- They differ in scope. Some **Android** devices allow users to modify or replace the operating system after unlocking the bootloader. Moreover, nearly all Android phones have an option to allow the user to install unknown, 3rd-party apps, so no exploit is needed for normal sideloading.
- **iOS** is engineered with security measures including a "locked bootloader" to prevent users from modifying the operating system, and to prevent apps from gaining root privileges. It violates Apple's end-user license agreement for iOS. Apps installed this way have the restrictions of all other apps. In addition, alternative app stores utilising enterprise certificates have sprung up, offering modified or pirated releases of popular iOS applications and video games, some of which were either previously released through Cydia or are unavailable on the App Store due to them not complying with Apple developer guidelines.

3. Types of Jailbreaking

Hacking iOS

- When a device is booting, it starts with loading the Apple kernel initially. The device must then be exploited and have the kernel patched each time it is turned on.
- An **"untethered" jailbreak** is a process where a jailbreak is achieved without the need to use a computer. As the user turns the device off and back on, the device starts up completely, and the kernel is patched.
- With a **"tethered" jailbreak**, a computer is needed to turn the device on each time it is rebooted. If the device starts back up on its own, it will not have a patched kernel. The purpose of the computer is to "re-jailbreak" the phone each time it is turned on.
- There is also a third kind called a **"semi-tethered"** solution. What this essentially means is that when the device boots, it will no longer have a patched kernel, but it can be used for normal functions.

Mobile Security Guidelines and tools

Module 18

1. Securing Android devices

Module 18

Mobile Security Guidelines and tools

- Delete invasive Android apps that abuse your privacy.
- Setup a VPN on your Android device to encrypt internet traffic.
- Block ads on your Android.
- Secure your SMS messages through encryption.
- Adjust your Android settings for more privacy and security.
- Turn on Google's malware scanner called Play Protect for Play Store apps.
- Turn on 2-step verification

Mobile Security Guidelines and tools

- Download a password manager
- Turn off connections when you don't need them. (BlueBorne)
- Use Lockdown mode
- Stop disclosing your location.
- Install Find My Device
- Prevent unknown downloads
- Check app permissions
- Always have full backups

2. Securing iOS Devices

Module 18

Mobile Security Guidelines and tools

- Create an iPhone Passcode (strong one)
- Use Touch ID or Face ID on iPhone
- Enable 'Find My iPhone'
- Control Your iPhone Privacy Settings
- Don't Jailbreak Your iPhone
- Make Encrypted iPhone Backups
- Use Security Apps on Your iPhone (VPN, Password Manager, etc.)
- Turn on two-step verification for Apple ID and iCloud
- Disable Siri on a lock screen
- Turn off automatic sync to iCloud

Mobile Security Guidelines and tools

- Discard automatic WiFi connections to known networks
- Turn off cookies in your browsers
- Turn off the AutoFill option in your browsers
- Don’t let apps access your contacts, photos, messages and other private data
- Make sure automatic iOS updates are turned on
- Change your reused passwords
- Turn on USB Restricted Mode to make hacking more difficult
- Don't share location data in images

3. Mobile Security Tools

Module 18

Mobile Security Guidelines and tools

- ImmuniWeb® MobileSuite
- Appvigil
- Ostorlab
- Zed Attack Proxy
- Kiuwan
- QARK
- Micro Focus
- Android Debug Bridge

Mobile Security Guidelines and tools

- AndroTotal
- CodifiedSecurity
- Drozer
- WhiteHat Security
- Synopsys
- Veracode
- SandDroid
- Mobile Security Framework (MobSF)

HACKING

Is an art, practised through a creative mind.