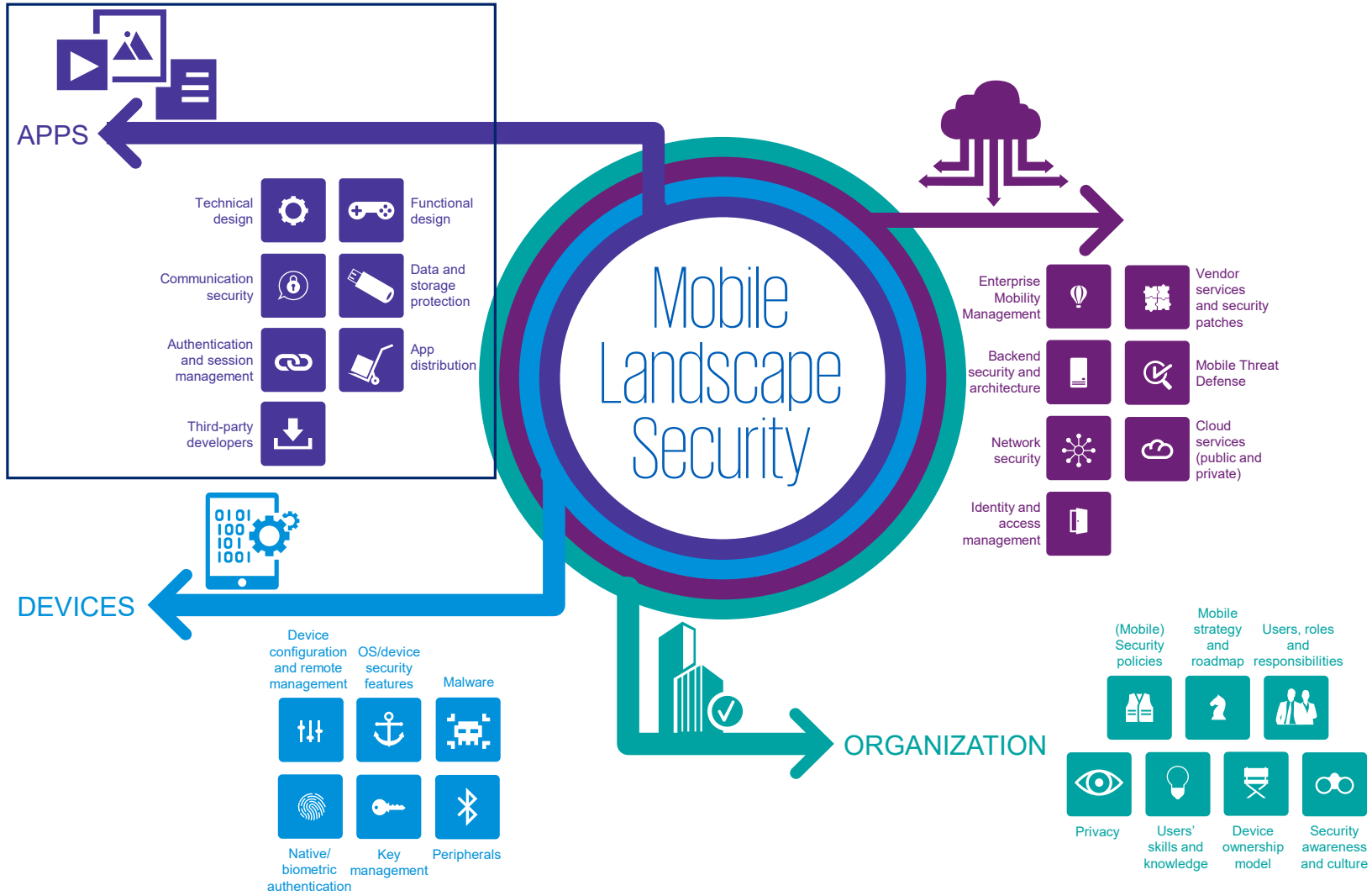
KPMG

Android Security Workshop - ECSC

Topics to discuss today

1. **Mobile security overview**
2. **Android Platform Overview**
3. **Android Hacking Fundamentals**
4. **Application code**
5. **App communication**
6. **Server-side (API & Infra)**
7. **Conclusion**

Mobile security and app security overview



KPMG

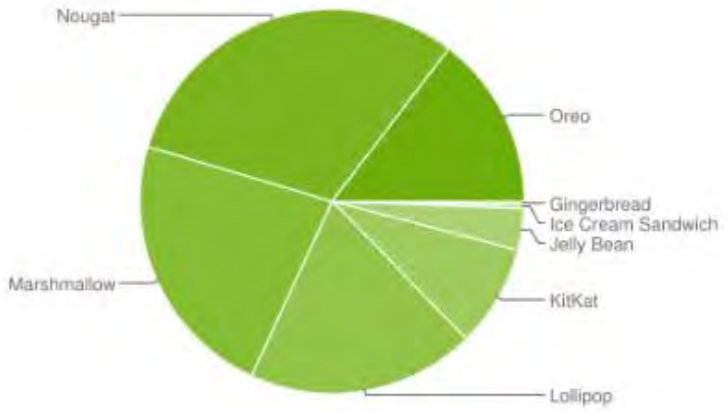
Android Platform Overview

Android Platform Overview

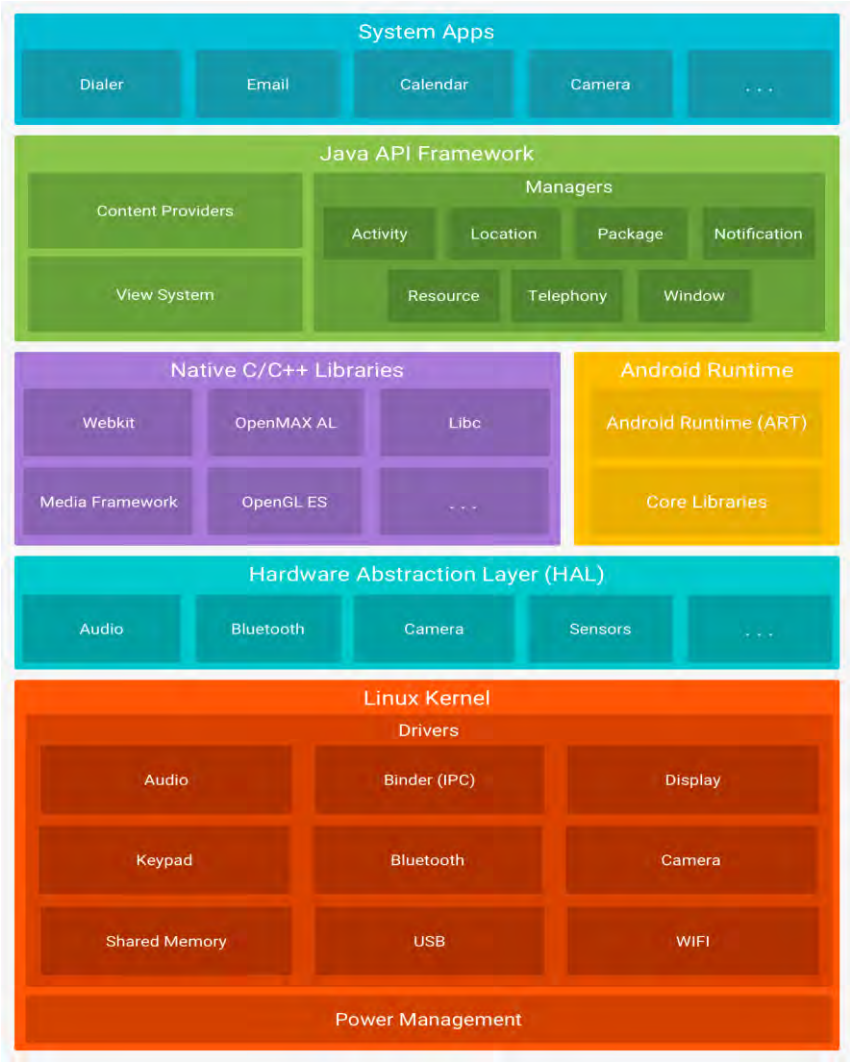
- Mobile OS developed by Google
- Operating system based on Linux Kernel
- Each app gets a unique Linux UID
- Market share ~75%
- Latest release: Android 9.0 (Pie)
- Apps are distributed via Google Play (vetted by Google) and other 3rd party stores (e.g. GetJar)

Android Distribution

Version	Codename	API	Distribution
2.3.3 - 2.3.7	Gingerbread	10	0.3%
4.0.3 - 4.0.4	Ice Cream Sandwich	15	0.3%
4.1.x	Jelly Bean	16	1.2%
4.2.x		17	1.8%
4.3		18	0.5%
4.4	KitKat	19	8.6%
5.0	Lollipop	21	3.8%
5.1		22	15.4%
6.0	Marshmallow	23	22.7%
7.0	Nougat	24	20.3%
7.1		25	10.5%
8.0	Oreo	26	11.4%
8.1		27	3.2%

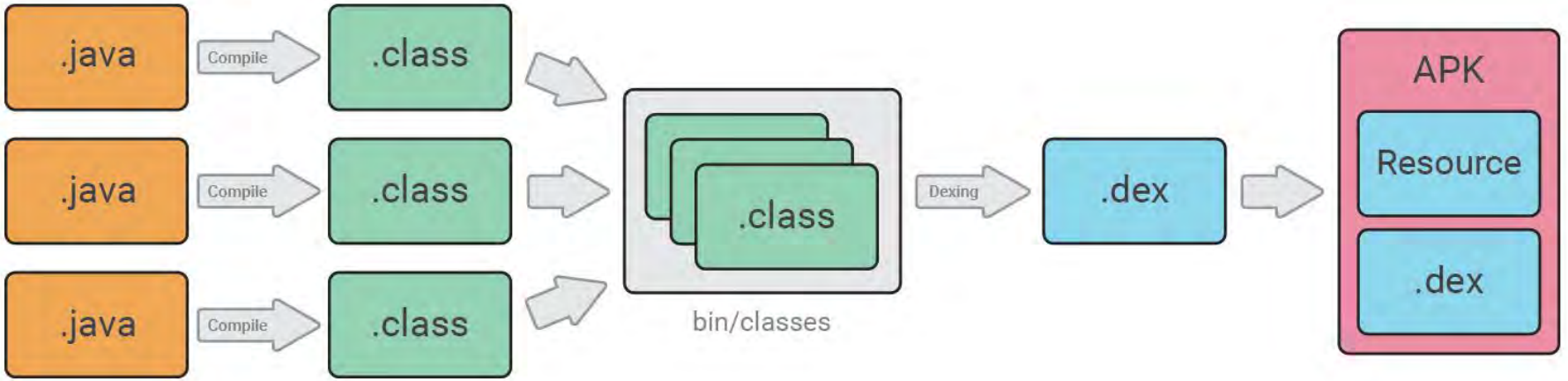


Android Architecture



Android Apps

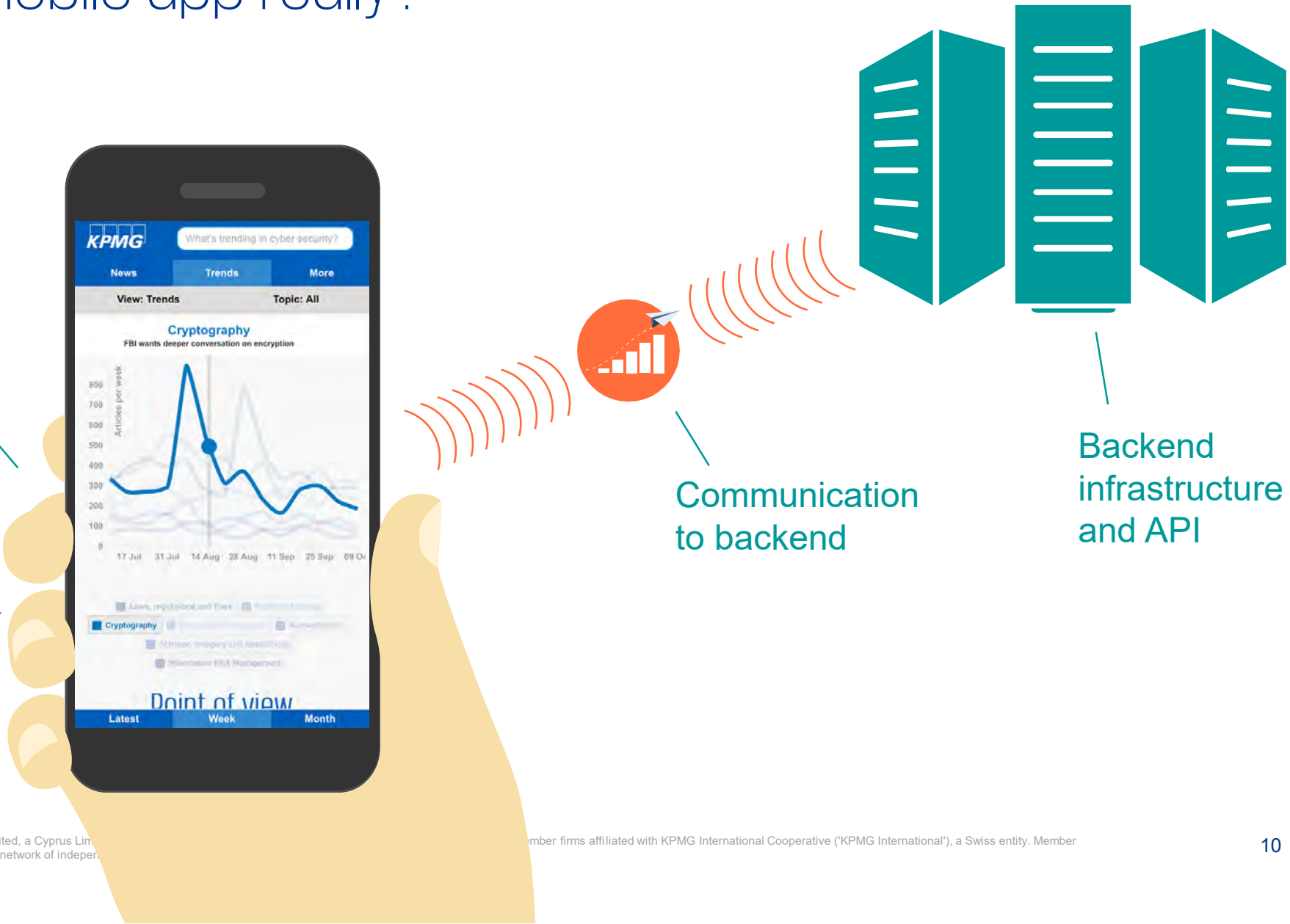
- Written in Java using the Android Native Development Kit (NDK)
- Android Manifest XML
- Apps are compiled into .dex (Dalvik Executable) bytecode



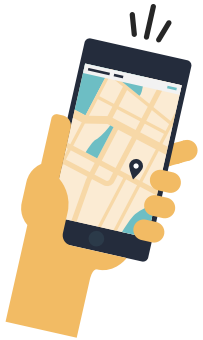
KPMG

Android App Hacking Fundamentals

What is a mobile app really?

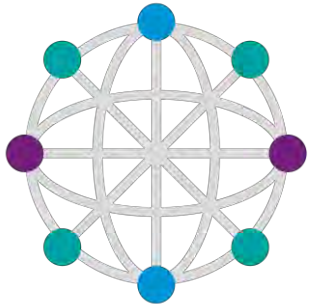


Components of App Security



App Code

- Local storage
- Interaction with other apps
- App behavior modification



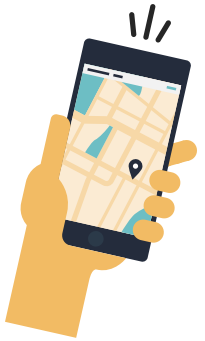
Communication

- TLS vulnerabilities
- Certificate pining
- E2E encryption (e.g. chat apps)

Server-side

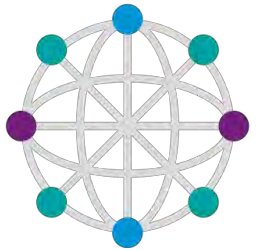
- Authentication & authorization
- Business Logic
- Web based attacks (injections, Session Mgmt etc.)
- Misconfigurations on the server

Application Code



- Static and Dynamic analysis (identification & exploitation)
- Reverse engineering the APK (check if the code is obfuscated)
- Vulnerable app components (e.g. broadcast receivers, intents or content providers)
- Sensitive data stored in memory/storage (even external storage!) or logs
- Misuse of Android permissions (privacy risk for the users)
- Root Detection/Prevention (check the code to find the flag)

Communication



- Check for certificate validation/pining
- TLS related vulnerabilities (Lucky13, Beast, Heartbleed etc.)
- End-to-end encryption becomes popular
- Information Disclosure (personal or other sensitive data is sent to 3rd parties)

Server-side (API & Infra)

- Insecure session management
  - Authentication & Authorization – malicious user can possibly bypass or manipulate these processes
- Parameter tampering which can lead to information leakage, privilege escalation or unauthorized access to data
- Business logic bypass (e.g. 2FA circumvention)
- Misconfigurations or vulnerabilities on the application server

App Components

- **Activities** – all the views (pages) of an app
- **Services** – Background long-running processes with no UI (e.g. play music)
- **Broadcast Receivers** – Listening for broadcast messages from other apps
- **Intents** – A mean of communication between apps' components (start activities, services or delivering broadcasts)
- **Content Providers** – Used to share data with other apps
- **Permissions** – Requested by the developer to access certain sensors/data (e.g. camera, internal/external storage etc.)

AndroidManifest.xml

OWASP mobile top 10 (2016)

Android Hacking Tools

drozer

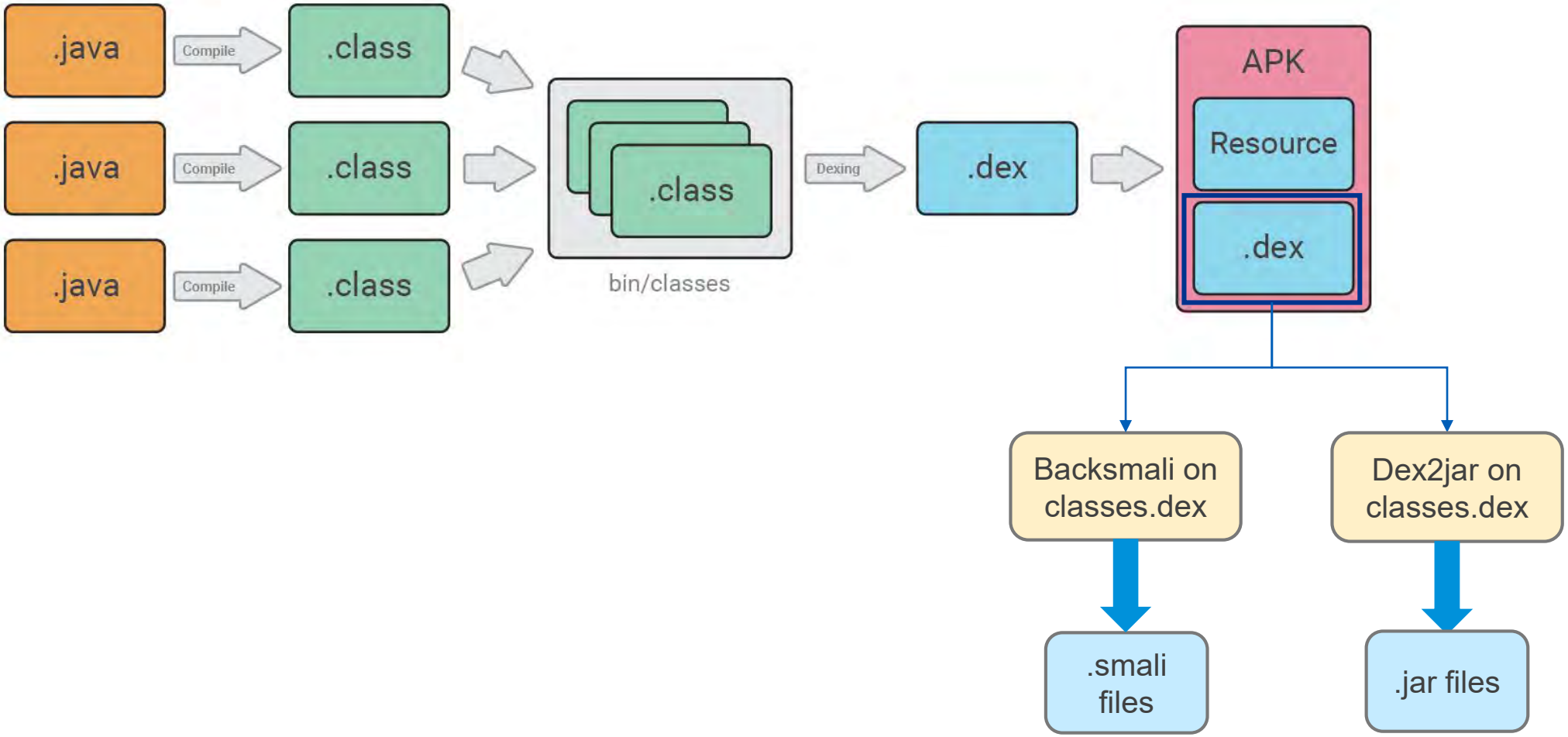
Dex2jar

```
 .d88888b.          d8888   8888888b.    888    d8P
d88P" "Y88b        d88888   888   Y88b   888   d8P
888     888       d88P888   888    888   888  d8P
888     888      d88P 888   888   d88P   888d88K
888     888     d88P  888   8888888P"    8888888b
888 Y8b 888    d88P   888   888 T88b     888  Y88b
Y88b.Y8b88P   d8888888888   888  T88b    888   Y88b
 "Y888888"   d88P     888   888   T88b   888    Y88b
       Y8b
```

KPMG

Application Code

Reversing an APK



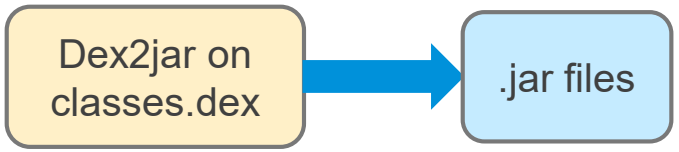
Reversing an APK

- Tool: APKTool
- Run the following command to decompile:
  `apktool d app.apk`
- The results will be:
  - AndroidManifest.xml (The most important file in Android)
  - Apktool.yml (essential file to rebuild the app)
  - /smali
  - /res
  - /original (essential to rebuild the app)

Example of Smali code

```
    .line 170
    .local v0, "i":Landroid/content/Intent;
    const-string v1, "passed_username"

    iget-object v2, p0, Lcom/android/insecurebankv2/LoginActivity;->Username_Text:Landroid/widget/EditText;

    invoke-virtual {v2}, Landroid/widget/EditText;->getText()Landroid/text/Editable;

    move-result-object v2

    invoke-virtual {v2}, Ljava/lang/Object;->toString()Ljava/lang/String;

    move-result-object v2

    invoke-virtual {v0, v1, v2}, Landroid/content/Intent;->putExtra(Ljava/lang/String;Ljava/lang/String;)Landroid/content/Intent;

    .line 171
    const-string v1, "passed_password"

    iget-object v2, p0, Lcom/android/insecurebankv2/LoginActivity;->Password_Text:Landroid/widget/EditText;

    invoke-virtual {v2}, Landroid/widget/EditText;->getText()Landroid/text/Editable;

    move-result-object v2

    invoke-virtual {v2}, Ljava/lang/Object;->toString()Ljava/lang/String;

    move-result-object v2

    invoke-virtual {v0, v1, v2}, Landroid/content/Intent;->putExtra(Ljava/lang/String;Ljava/lang/String;)Landroid/content/Intent;

    .line 172
    invoke-virtual {p0, v0}, Lcom/android/insecurebankv2/LoginActivity;->startActivity(Landroid/content/Intent;)V

    .line 173
    return-void
.end method
```

Go back to Java



- Tool: Dex2jar (converts .dex to .jar)
- We first need to unzip the .apk file and then run:

  `d2j-dex2jar classes.dex`
- The result will be a .jar file which contains all the java classes
- Now static code analysis is much easier
- jd-gui is then used to display the .jar file

Example of Java code

```
protected void performlogin()
{
  this.Username_Text = ((EditText)findViewById(2131165251));
  this.Password_Text = ((EditText)findViewById(2131165250));
  Intent localIntent = new Intent(this, DoLogin.class);
  localIntent.putExtra("passed_username", this.Username_Text.getText().toString());
  localIntent.putExtra("passed_password", this.Password_Text.getText().toString());
  startActivity(localIntent);
}
```

Code Obfuscation

Original Code

```
public class test1 {
  private int term1;
  private int term2;
  private boolean areRelativelyPrime;

  public test1(int term1, int term2){
    this.term1 = term1;
    this.term2 = term2;
    areRelativelyPrime =
      areRelativelyPrime();
  }

  private static int
        gcd(int term1, int term2) {
    int remainder;

    remainder = term1 % term2;
    if (remainder == 0) {
      return term2;
    }
    else {
      return gcd(term2, remainder);
    }
  }


  private boolean
    areRelativelyPrime()
    {if (gcd(term1, term2) == 1) {
      return true;
    }
    else {
      return false;
    }
  }
}
```

Obfuscated Code

```
public class a {
  private int a;
  private int b;
  private boolean c;

  public a(int a, int b) {
    this.a = a;
    this.b = b;
    c = c();
  }


  private static int b(int a, int b) {

    int c;

    c = a % b;
    if (c == 0) {
      return b;
    }
    else {
      return b(b, c);
    }
  }


  private boolean c() {

    if (b(a, b) == 1) {
      return true;
    }
    else {
      return false;
    }
  }
}
```

Check for vulnerabilities

- Inspect the AndroidManifest.xml to identify:
  - Exported activities, broadcast receivers or services
  - Excessive permissions requested by the author
  - MinSDK version which the app will run on (if too old, too bad)
  - If the app is debuggable – runtime code injection
  - If the app can be backed-up

Check for vulnerabilities

- Inspect the local storage (internal/external):
  - App data are stored in `/data/data/<app-name>/`
  - Local databases (SQLite), shared preferences and cache
  - External storage which is either:
    - The physical SD card of the user or
    - The emulated one that the Android OS creates at `/sdcard` or `/mnt/sdcard`

Check for vulnerabilities

- Inspect the code for:
  - Identification of vulnerable components
  - Possible hardcoded secrets (e.g. admin, password, encryption keys etc.)
  - Sensitive data stored in log files, sent to obscure servers (e.g. 3rd parties)
  - Possible SQL injections, command executions

Exploitation

- Exploitation of any vulnerabilities identified in static analysis by using ADB or Drozer
- Basic adb Commands:

```
adb shell //Get a shell on the device
adb logcat //Show the Android system logs
adb shell am start //Start an activity via an intent
adb shell am startservice //Start a service
adb shell am stopservice //Stop a service
adb shell am broadcast //Send a broadcast intent
adb shell content [insert,query,delete] //Send a command to a content provider
```

KPMG

App Communication

What does an app send?

Overwriting of information on the server

Payment data and personal information

Authentication & authorization data

Certificate Pinning

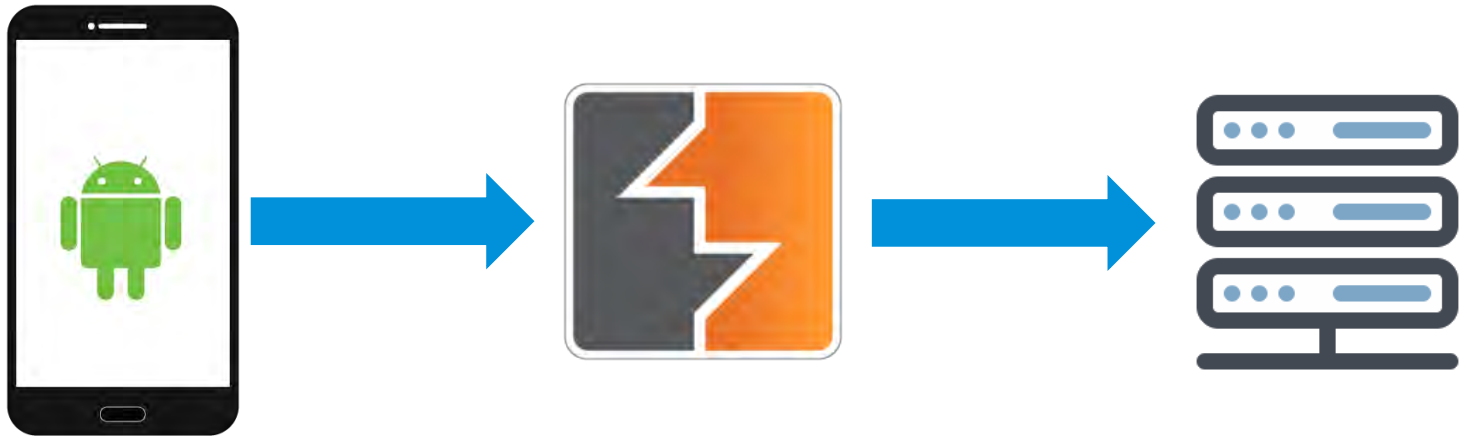
- SSL pinning is the process of checking if the server's certificate is exactly the certificate that you expect to be
- It is used to prevent almost all classes of man-in-the-middle (MiTM) attacks
- Since mobile applications should connect to a single or limited set of backends, almost all mobile apps should be expected to do certificate pinning

Certificate Pinning


```
TrustManager[] trustAllCerts = new TrustManager[] {
    new X509TrustManager() {
        @Override
        public X509Certificate[] getAcceptedIssuers() {
            return new java.security.cert.X509Certificate[] {};
        }

        @Override
        public void checkClientTrusted(X509Certificate[] chain, String authType)
            throws CertificateException {
        }

        @Override
        public void checkServerTrusted(X509Certificate[] chain, String authType)
            throws CertificateException {
        }
    }
 };

// SSLContext context
context.init(null, trustAllCerts, new SecureRandom());
```

Certificate Pinning - Bypass

- If certificate pinning is used, but the app is sufficiently reverse engineerable you can insert your own certificate by replacing the certificates and repacking the app



Considering all communication channels

- An app usually not only communicates with one backend
  - Consider multiple data sources an app communicates with
  - Consider information leaks in downloading assets from CDNs or different servers
  - Consider information sent in app analytics.

KPMG

Server-side (API & Infra)

API attacks

- Insecure authentication
  - Weak password policies
  - Bypass authentication and access restricted recourses directly
  - Session token – is it random (cryptographically secure)?
- Insecure authorization
  - Access other users' data
  - Perform actions that were not indented for your role/user group
- Lack of input validation

API attacks

- Business logic flaws
  - Bypass a security feature (e.g. 2FA) or generate a discount for a product that was not supposed to have a discount
  - Manipulate a cookie to become admin or other user
  - Exploit file upload weaknesses (e.g. when submitting a photo in an app)
  - Change the normal flow of an application to circumvent a security feature

Infrastructure vulnerabilities

- Unpatched systems/software
- Default credentials on an exposed administration panel
- Default configurations (no hardening)
- Services exposed externally
- Misconfiguration of software/systems

Conclusion

In Summary

1. **Mobile security overview**
2. **Android Platform Overview**
3. **Android Hacking Fundamentals**
4. **Application code**
5. **App communication**
6. **Server-side (API & Infra)**
7. **Conclusion**

Vulnerable apps to play with

- OWASP GoatDroid
- Damn Vulnerable Hybrid App
- Damn Insecure Vulnerable Application (DIVA)
- InsecureBankv2 (you will play with this today)
- …

Thank you