WILLIAM & MARY

CHARTERED 1693

CSCI 445:
Mobile Application Security

Lecture 10

Prof. Adwait Nadkarni

Homework 2 (discussion)

- Great job folks!
- Average score ~43!
- *5/35 scored 50/50!*

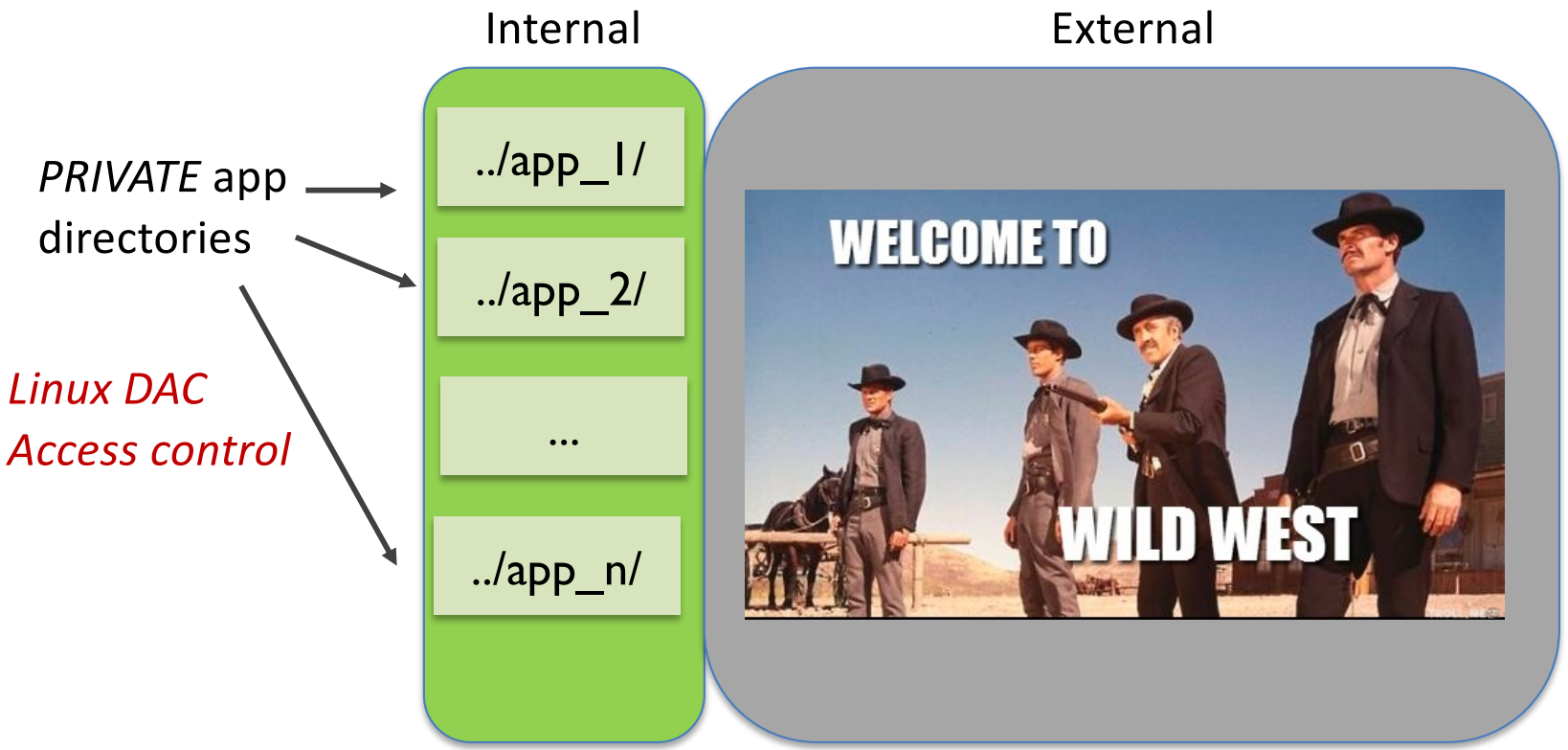
HW3 assigned

- Due on March 19th, 11:59PM (*email for extension*)
- *Basic* Application analysis assignment: *Manual analysis*
  - Input: Application with vulnerabilities of two types:
    - SSL misuse
    - Exported component(s), from last class
  - **Task:** Find and describe the specific vulnerabilities (i.e., the lines of vulnerable code)
  - **lastname-hw3.pdf:** Report containing the identified vulnerabilities. **Use LaTeX.**
  - **Extra Credit:** Recommend correct fixes for *all* the vulnerabilities, get 1 bonus on the course grade.

Android's storage architecture

App Storage on Android

Q: *Why do apps need external storage (sdcard)?*



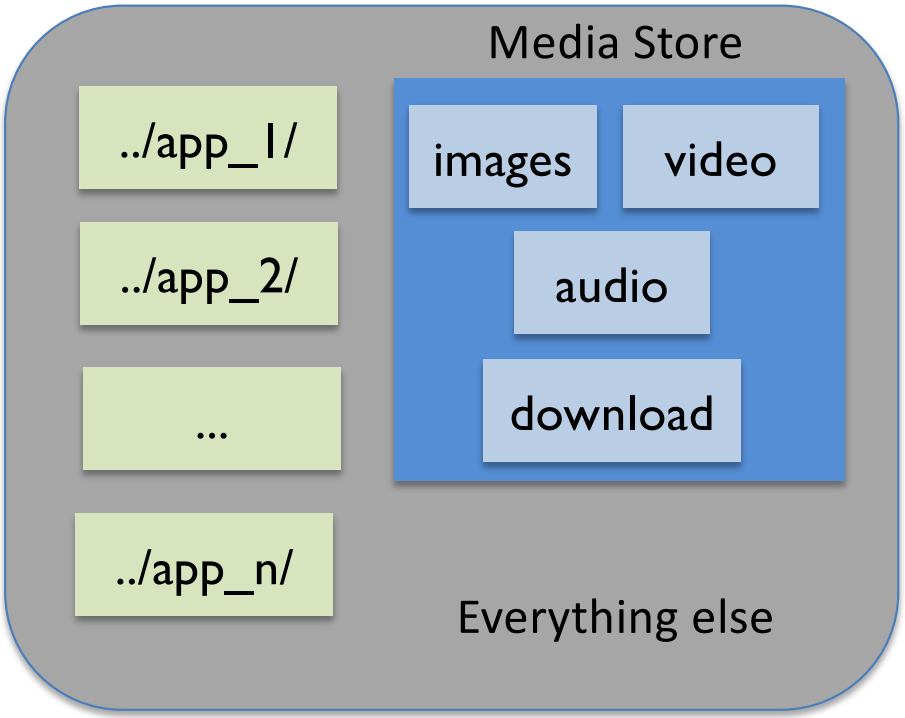
Any app with READ/WRITE_EXTERNAL STORAGE can access all data! (but this is changing)

Towards better external storage

- Apps need to read/write files on external storage
  - How can we facilitate access without abuse?
    - *Do all apps need **broad** storage access?*
- Two kinds of apps:
  1. Apps that want to offload their data
  2. Apps that want to *manage* data (e.g., file managers)
     - Insight: *Only the latter need broad storage access!*

External storage (now)

Apps need no permission to access their dirs



Apps need media-specific runtime permissions

- Apps with READ/WRITE_EXTERNAL_STORAGE permissions can still access everything. *So how does this new design help?*
- *Apps that do not need broad storage access can function with none or media-specific permissions!*

External Storage (future)

- *Scoped Storage* all the way!
  - Apps will *only* be able to access their own private directories on external storage
  - Read/write external storage permissions will not be widely granted
- *What about file managers?*
  - *Apps will need to specially request Google for broad storage permissions*
- *Does this solve the problem of abuse of broad storage access?*

Protecting Data at Rest

Storing Sensitive Data (1)

- Applications often require the use of sensitive data such as user credentials
- The attacker may try to access this data on storage.
- *Option 1:* Do not store it on the device

Storing Sensitive Data (2)

- Applications often require the use of sensitive data such as user credentials
- *Option 2:* Store it in the app's private directory
  - /data/data/<app-package-name>

```
//API to read/write files in the PRIVATE path, i.e.,/data/data/<app_packagename>/files/
FileInputStream fis = openFileInput("input.txt");

//Note: NEVER use MODE_WORLD_READABLE/WRITABLE unless there is no other option.
FileOutputStream fos = openFileOutput("output.txt", MODE_PRIVATE);
int ch;
while((ch = fis.read())!=-1){
      fos.write(ch);
}
```

Other Useful APIs:
getFilesDir(), getDir(), deleteFile(), fileList()

Thankfully deprecated in Android N, but still found in some apps

Storing Sensitive Data (3)

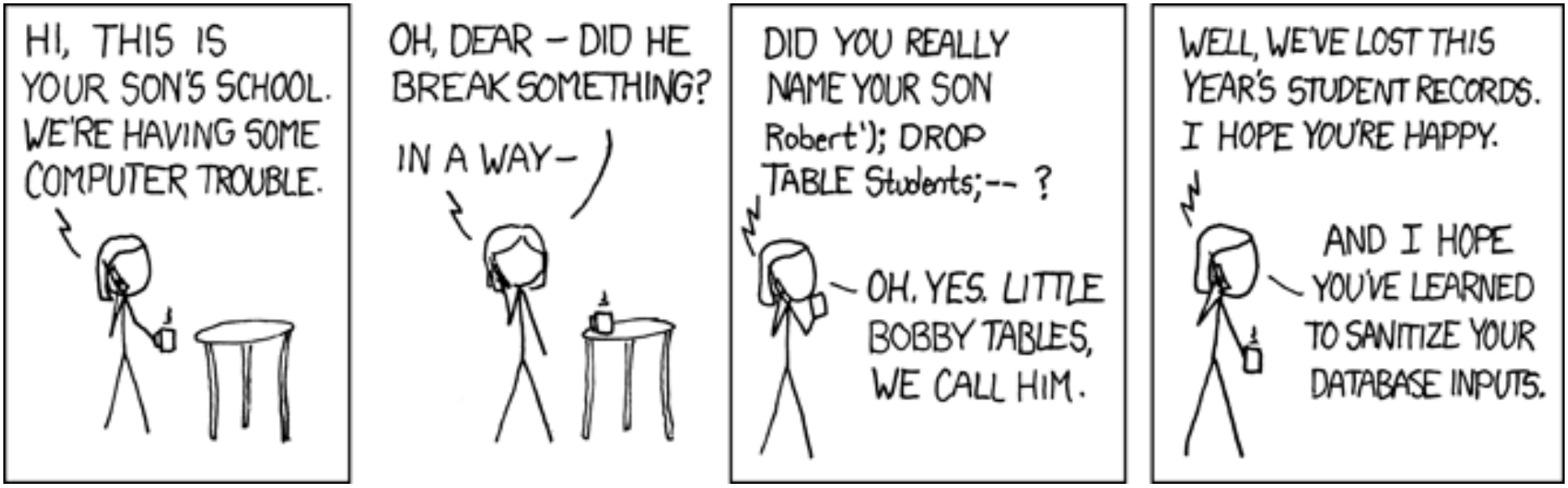
- Applications often require the use of sensitive data such as user credentials
- *Option 3*: Encrypt it!
  - Use the *Android KeyStore* for key generation and storage

```
KeyPair generateKeys() throws Exception {
    Calendar cal = Calendar.getInstance();
    Date now = cal.getTime();
    cal.add(Calendar.YEAR, 1);
    Date end = cal.getTime();

    KeyPairGenerator kpg = KeyPairGenerator.getInstance("RSA", "AndroidKeyStore");
    kpg.initialize(new KeyPairGeneratorSpec.Builder(context).setAlias(alias)
                        .setStartDate(now).setEndDate(end)
                        .setSerialNumber(BigInteger.valueOf(1))
                        .setSubject(new X500Principal("CN=SampleCN")).build());

    return kpg.generateKeyPair();
}
```

And finally...


HI, THIS IS YOUR SON'S SCHOOL. WE'RE HAVING SOME COMPUTER TROUBLE.
OH, DEAR – DID HE BREAK SOMETHING?
IN A WAY–
DID YOU REALLY NAME YOUR SON Robert'); DROP TABLE Students;-- ?
OH. YES. LITTLE BOBBY TABLES, WE CALL HIM.
WELL, WE'VE LOST THIS YEAR'S STUDENT RECORDS. I HOPE YOU'RE HAPPY.
AND I HOPE YOU'VE LEARNED TO SANITIZE YOUR DATABASE INPUTS.

Prevent SQL Injection

- *SQL injection*: user provides a substring for an SQL query that changes the query entirely (e.g., add SQL operations to query processing)

```
SELECT fieldlist FROM table
 WHERE field = 'anything' OR 'x'='x';
```

```
SELECT UserId, Name, Password FROM Users WHERE UserId = 105 or 1=1;
```

- Use *parameterized SQL* methods for the implementation of query, insert, etc.

```
// Partial Implementation of the Content Provider's query method.
public Cursor query(Uri uri, String[] projection, String selection, String[] selectionArgs) {
    // Query the underlying database using the SQLiteDatabase query method,
    //Instead of calling rawQuery(String sqlQuery, String[] selectionArgs);
    SQLiteDatabase db = dbHelper.getWritableDatabase();
    SQLiteQueryBuilder qb = new SQLiteQueryBuilder();
    return qb.query(db, projection, selection, selectionArgs, null, null, orderBy);
}
```

Takeaways

1

Use default APIs

- getFilesDir()/getExternalFilesDir()
- MediaStore APIs

2

Move to scoped storage, adhere to least privilege

3

Encrypt credentials w/ Keystore