

Web Communication Analysis of Android Applications

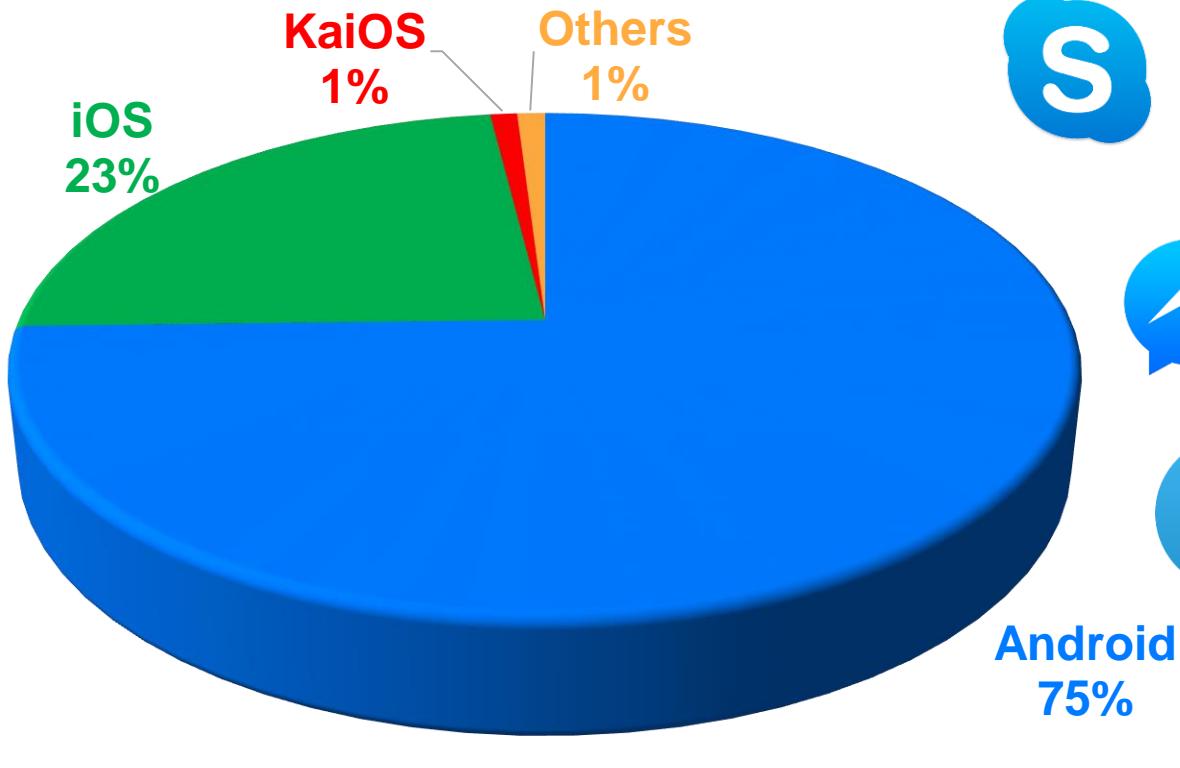
Master Thesis - Final Presentation

u^b

b
**UNIVERSITÄT
BERN**

Marc-Andrea Tarnutzer
09.04.2019

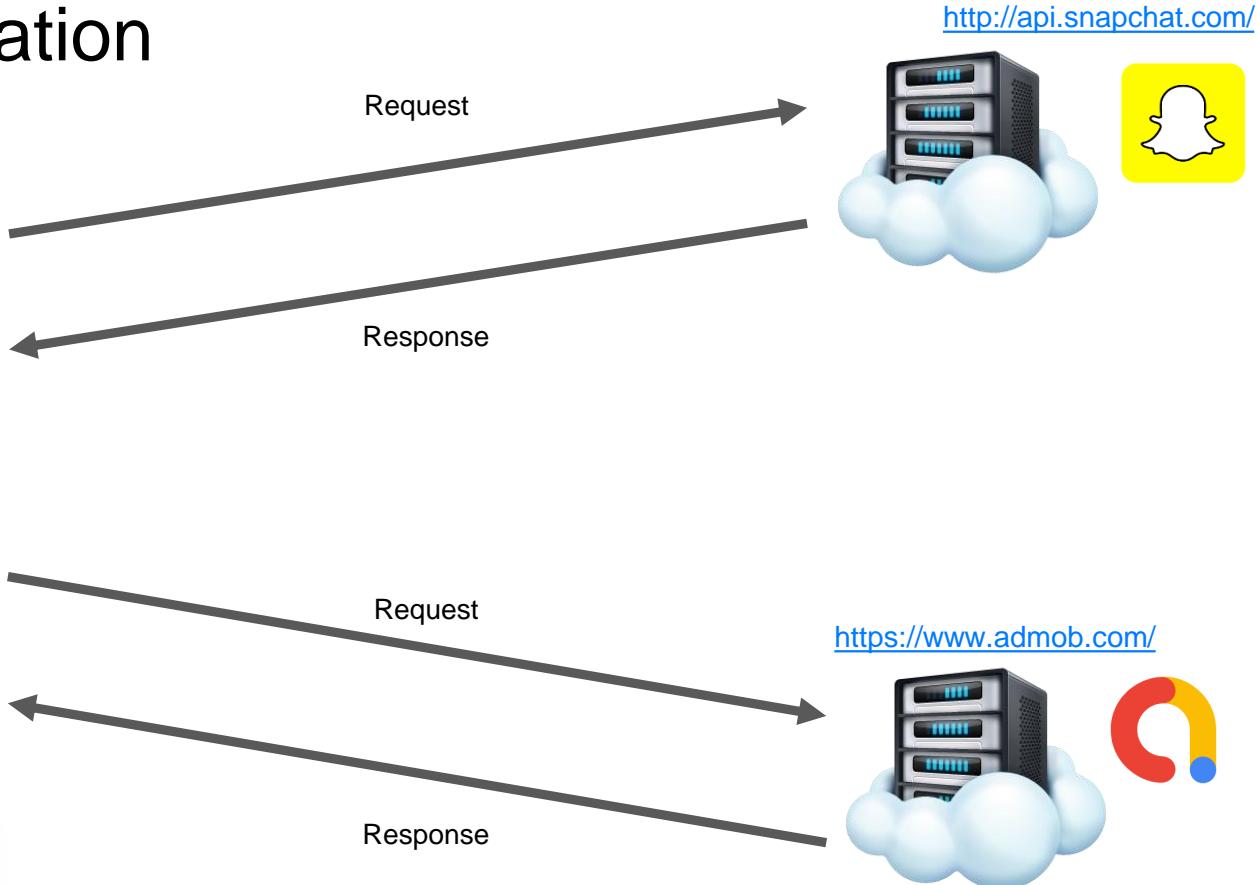
Networked Appified World



Android
75%



Web Communication



Research Questions

RQ1) Prevalent web communication frameworks?

RQ2) Transmitted data?

RQ3) Differences open-source vs closed-source apps?

RQ4) Web server configurations?

Web Communication

Background

Application server(s)

Application Programming Interface (API)



Request

API endpoint
(server)

Response



REpresentational State Transfer (REST)

Client-Server Architecture

Layered System

Stateless

Cacheable

Uniform interfaces

Code on demand

Data Schemes

```
{  
    "id": 3948292,  
    "name": "Miller",  
    "first_name": "Bob",  
    "address": {  
        "street": "Sample Street 1",  
        "number": 10  
    },  
    "married": true,  
    "badges": [  
        "badge1",  
        "badge2" ],  
    "social_media": null  
}
```



```
<?xml version="1.0" encoding="UTF-8"?>  
<root>  
    <id>3948292</id>  
    <name>Miller</name>  
    <first_name>Bob</first_name>  
    <address>  
        <street>Sample Street 1</street>  
        <number>10</number>  
    </address>  
    <married>true</married>  
    <badges>badge1</badges>  
    <badges>badge2</badges>  
    <social_media/>  
</root>
```



Common HTTP Request Methods

GET

Retrieve resource

`www.example.com/api/user?id=42`

POST

Create resource

HTTP body: `{ "id": 42, "name": "Bob Smith", ... }`

PUT

Replace resource

HTTP body: `{"id": 42, "name": "Adam Smith", ... }`

PATCH

Modify resource

HTTP body: `{ "age": 33 }`

DELETE

Delete resource

`www.example.com/api/user?id=42`

Simplified HTTP Request

- 1 Request line
- 2 Request headers
- 3 Request message body
- 4 Server status
- 5 Response headers
- 6 Response message body

```
PATCH /api/user/20382 HTTP/1.1 1
Host: someservice.com 2
Accept: application/json
Accept-Encoding: application/gzip
cache-control: no-cache

{
  "user_name": "Bob" 3
}
```

```
HTTP 200 No Error 4
Content-Type: application/json; charset=UTF-8
Access-Control-Allow-Origin: *
Pragma: no-cache
Set-Cookie: PHPSESSID=u45n2me6f0jueflptpjmrvid4; path=/
Server: nginx/1.14.1
Access-Control-Expose-Headers: Content-Type, X-Requested-With, X-authentication, X-client
Transfer-Encoding: Identity
Referrer-Policy:
Expires: Thu, 19 Nov 1981 08:52:00 GMT
Cache-Control: no-store, no-cache, must-revalidate, post-check=0, pre-check=0
Date: Sun, 31 Mar 2019 15:31:41 GMT
Connection: keep-alive
Vary: Accept-Encoding

{
  "success": true 5
}
```

Request

Response

RQ1:

Which are the *prevalent web communication frameworks* used in mobile apps?

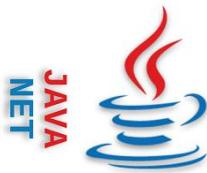
Found Web Communication Facilities

(manual analysis of 160 apps)

HTTP libraries



*Android
Volley*



OkHttp



glide

 **Retrofit**



AndroidHttpClient

**AsyncHttpClient
(loopj)**

Ion

JSON libraries

Gson

github.com/google/gson

org.json

[Included in Android](#)

Moshi

github.com/square/moshi

REST Library Support



```
URL url = new URL("http://www.google.ch");
Socket socket = new Socket(url.getHost(), 80);
OutputStream outputStream = socket.getOutputStream();
PrintWriter printWriter = new PrintWriter(outputStream, false);
printWriter.print("GET / HTTP/1.1\r\n");
printWriter.print("Host: www.google.ch\r\n");
printWriter.print("Connection: Close\r\n");
printWriter.print("\r\n");
printWriter.flush();
InputStream inputStream = socket.getInputStream();
InputStreamReader inputStreamReader = new InputStreamReader(inputStream);
BufferedReader bufferedReader = new BufferedReader((InputStreamReader));
int in;
while ((in = bufferedReader.read()) != -1) {
    System.out.print((char) in);
}
bufferedReader.close();
```

without REST library



```
String url = "http://www.google.com";
StringRequest stringRequest = new StringRequest(Request.Method.GET, url,
    new Response.Listener<String>() {
        @Override
        public void onResponse(String response) {
            System.out.print("Response: " + response);
        }
    }, new Response.ErrorListener() {
        @Override
        public void onErrorResponse(VolleyError error) {
            System.out.println("Error: " + error);
        }
});
```

with REST library

JSON Library Support



```
String jsonString = "{\"k1\":\"v1\", " +  
    "\"k2\":null, " +  
    "\"k3\":9, " +  
    "\"k4\":{\"k5\":\"v5\"}}";
```

without JSON library



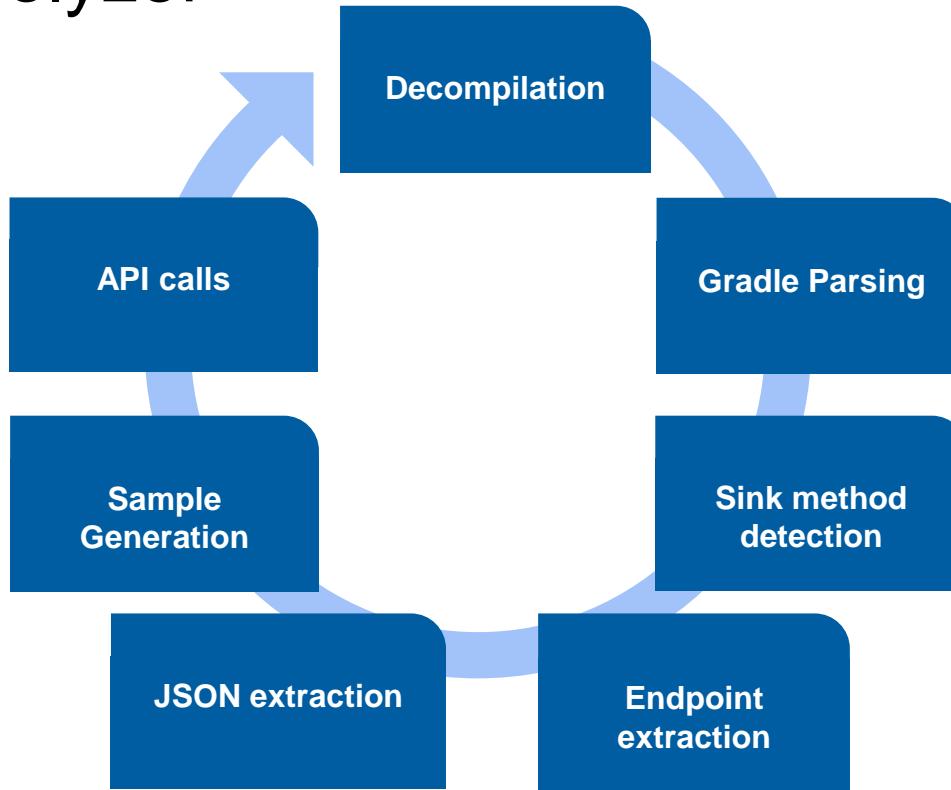
```
JSONObject jsonObject = new JSONObject();  
try {  
    jsonObject.put("k1", "v1");  
    jsonObject.put("k2", JSONObject.NULL);  
    jsonObject.put("k3", 9);  
    JSONObject jsonObject2 = new JSONObject();  
    jsonObject2.put("k5", "v5");  
    jsonObject.put("k4", jsonObject2);  
} catch (JSONException e) {  
    e.printStackTrace();  
}
```

with JSON library

Jandrolyzer

Introduction

Jandrolyzer



Java.net

OkHttp

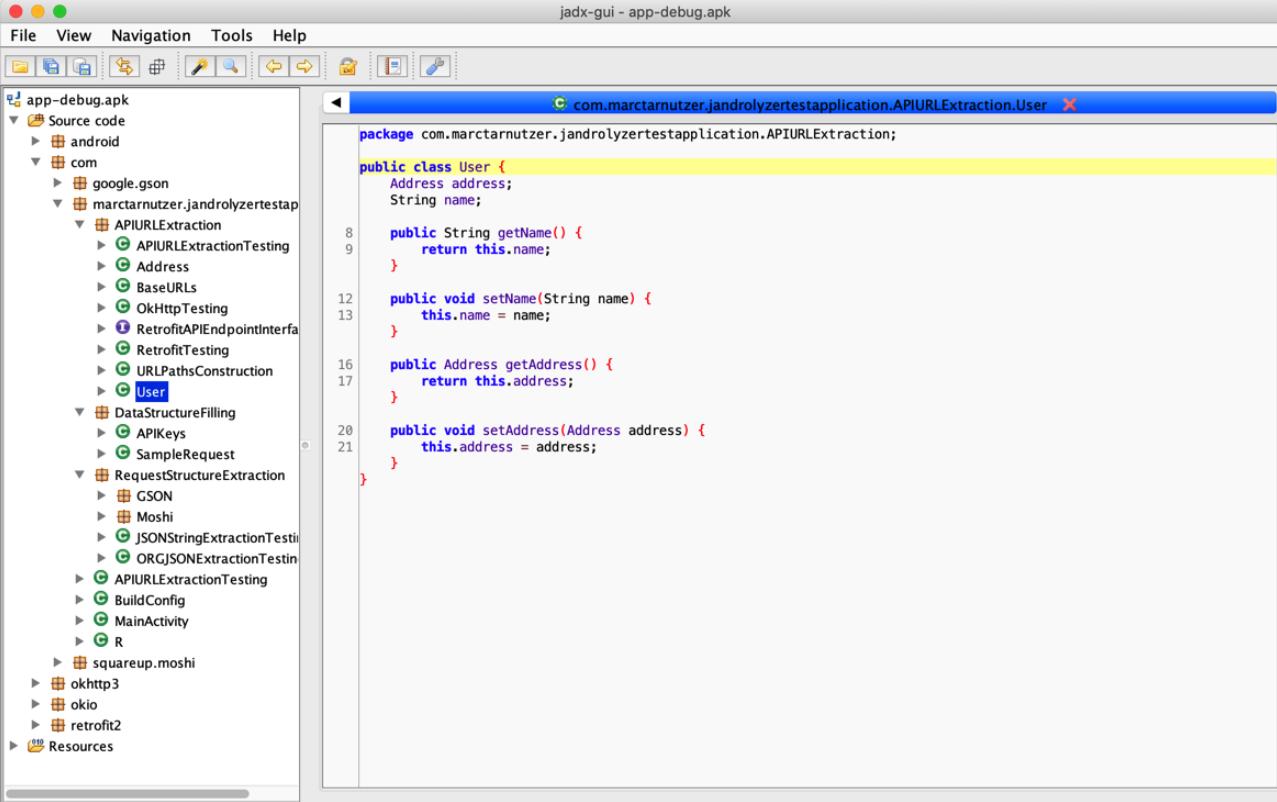
Retrofit

Gson

Moshi

org.json

Analysis Step 1 / 7: Decompilation



The screenshot shows the JADX GUI interface with the title "jadex-gui - app-debug.apk". The left sidebar displays the project structure under "app-debug.apk" with the "Source code" tab selected. The tree view shows packages like android, com, and marctarnutzer.jandrolyzertestapplication, along with sub-packages such as APIURLExtraction, DataStructureFilling, RequestStructureExtraction, and various utility classes like GSON, Moshi, and RetrofitTesting.

The main window displays the decompiled Java code for the `User` class:

```
package com.marctarnutzer.jandrolyzertestapplication.APIURLExtraction;

public class User {
    Address address;
    String name;

    public String getName() {
        return this.name;
    }

    public void setName(String name) {
        this.name = name;
    }

    public Address getAddress() {
        return this.address;
    }

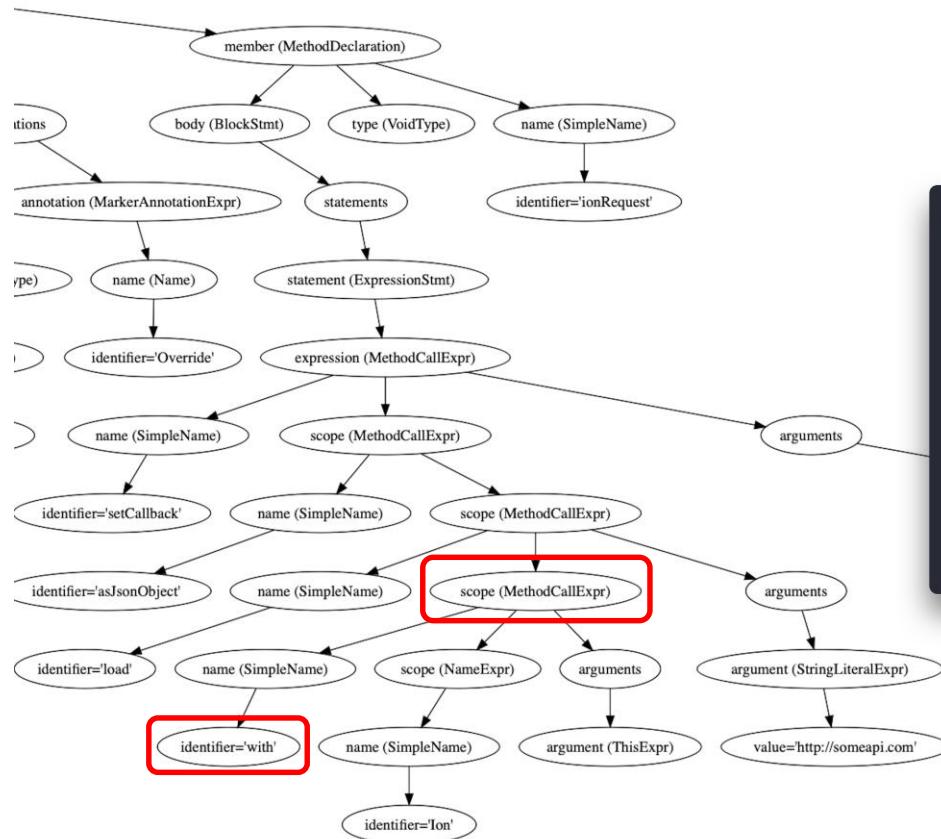
    public void setAddress(Address address) {
        this.address = address;
    }
}
```

Analysis Step 2 / 7: Gradle Parsing

The image shows a file browser on the left and a code editor on the right. The file browser displays the project structure of 'JandrolyzerTestApplication' with various files and folders like .gradle, .idea, app, build, libs, src, androidTest, main, java, res, and test. Specific files like 'AndroidManifest.xml', 'build.gradle', and 'gradlew' are highlighted with red boxes. The code editor on the right contains the build.gradle script, which defines the application's configuration, dependencies, and build types.

```
1 apply plugin: 'com.android.application'
2
3 android {
4     compileSdkVersion 26
5     buildToolsVersion "26.0.2"
6     defaultConfig {
7         applicationId "com.marctarnutzer.jandrolyzertestapplication"
8         minSdkVersion 15
9         targetSdkVersion 26
10        versionCode 1
11        versionName "1.0"
12        testInstrumentationRunner "android.support.test.runner.AndroidJUnitRunner"
13    }
14    buildTypes {
15        release {
16            minifyEnabled false
17            proguardFiles getDefaultProguardFile('proguard-android.txt'), 'proguard-rules.pro'
18        }
19    }
20}
21
22dependencies {
23    compile fileTree(dir: 'libs', include: ['*.jar'])
24    androidTestCompile('com.android.support.test.espresso:espresso-core:2.2.2', {
25        exclude group: 'com.android.support', module: 'support-annotations'
26    })
27    compile 'com.android.support:appcompat-v7:26.+'
28    compile 'com.android.support.constraint:constraint-layout:1.0.2'
29    testCompile 'junit:junit:4.12'
30
31    // OkHttp dependencies
32    implementation("com.squareup.okhttp3:okhttp:3.12.1")
33
34    // Retrofit dependencies
35    implementation 'com.squareup.retrofit2:retrofit:2.5.0'
36    implementation 'com.squareup.retrofit2:converter-gson:2.5.0'
37
38    // Moshi dependencies
39    implementation 'com.squareup.moshi:moshi:1.8.0'
40}
41
```

Analysis Step 3 / 7: Sink method detection



com.koushikdutta.ion.builder.LoadBuilder

A screenshot of a Java code editor showing the following code:

```
Ion with(this)
    .load("http://someapi.com")
    .asJsonObject()
    .setCallback(new FutureCallback<JsonObject>() {
        @Override
        public void onCompleted(Exception e, JsonObject result) {
            System.out.print("Result: " + result);
        }
    });

```

The code editor highlights two parts of the code with red boxes:

- The `with(this)` call is highlighted.
- The `setCallback` method call is highlighted.

Analysis Step 4 / 7: Endpoint Extraction

interface implementation

```
Retrofit retrofit = new Retrofit.Builder()  
    .baseUrl("http://retrofiturl.com/")  
    .addConverterFactory(GsonConverterFactory.create())  
    .build();  
  
RetrofitAPIEndpointInterface rApiInt  
    = retrofit.create(RetrofitAPIEndpointInterface.class);
```

http://oldapi.com/api/loadNewsFromOldAPI

http://retrofiturl.com/api/loadUser/<INTEGER>

http://retrofiturl.com/api/loadUsers

http://retrofiturl.com/api/createUser

interface declaration

```
public interface RetrofitAPIEndpointInterface {  
    @GET("http://oldapi.com/api/loadNewsFromOldAPI")  
    Call<Article> loadNewsFromOldAPI();  
  
    @GET("api/loadUser/{id}")  
    Call<User> loadUser(@Path("id") int userId);  
  
    @GET("api/loadUsers")  
    Call<List<User>> loadUsers(@Query("order") String order,  
                                @Query("position") String position);  
  
    @POST("api/createUser")  
    Call<User> createUser(@Body User user);  
}
```

Analysis Step 5 / 7: JSON Extraction

```
public void sampleReq() {  
    String id = "AAAABBBBCCCC";  
    updateUser(id);  
}  
  
private void updateUser(String id) {  
    JSONArray jsonArray = new JSONArray();  
    jsonArray.put("badge1");  
    jsonArray.put("badge2");  
  
    JSONObject jsonObject = new JSONObject();  
    try {  
        jsonObject.put("user_id", id);  
        jsonObject.put("VIP", true).put("score", 1);  
        jsonObject.put("badges", jsonArray);  
    } catch (JSONException e) {  
        e.printStackTrace();  
    }  
  
    ...  
}
```

```
{  
    "user_id": "AAAABBBBCCCC",  
    "VIP": true,  
    "score": 1,  
    "badges": [  
        "badge1",  
        "badge2"  
    ]  
}
```

Analysis Step 6 / 7: Sample Generation

extracted URLs & JSON

```
https://someservice.com/api/registerDevice?api_key=<STRING>  
  
{  
    "device_id": "AAAABBBBCCCC",  
    "api_key": "<STRING>"  
}
```

extracted variables

```
{"refreshToken": ["REFRESHTOKEN"]}  
{"accessToken": ["ACCESSTOKEN"]}  
{"apiKey": "APIKEY"}  
{"id": ["userId"]}  
  
...
```

Jaro-Winkler similarity: 0.8944...

final result

```
https://someservice.com/api/registerDevice?api_key=APIKEY  
  
{  
    "device_id": "AAAABBBBCCCC",  
    "api_key": "APIKEY"  
}
```

Analysis Step 7 / 7: API Calls



```
https://api.com/path  
https://api.com/path2  
http://someservice.com/some/path  
http://someservice.com/api/users  
http://otherservice.com/api/news
```

...



```
{"k":"v","k2":91,"k3":null,"k4":9}  
{"test":10}  
{"name":"test"}  
{"address":{"street":"sample street","number":118}, "name": "Bob"}
```

...



RQ2:

What ***data*** do mobile apps transmit through web communication channels?

RQ3:

What are the ***differences*** between open-source and closed-source apps in regard to web communication?

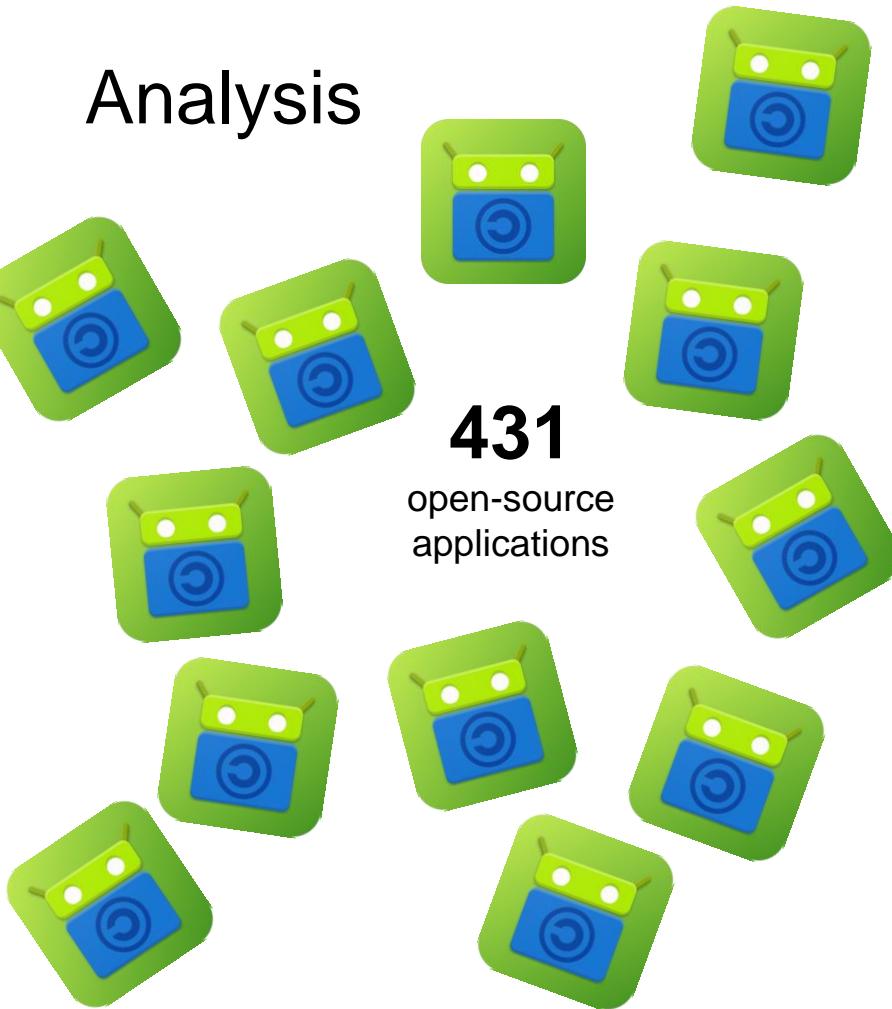
Jandrolyzer

Results

Analysis

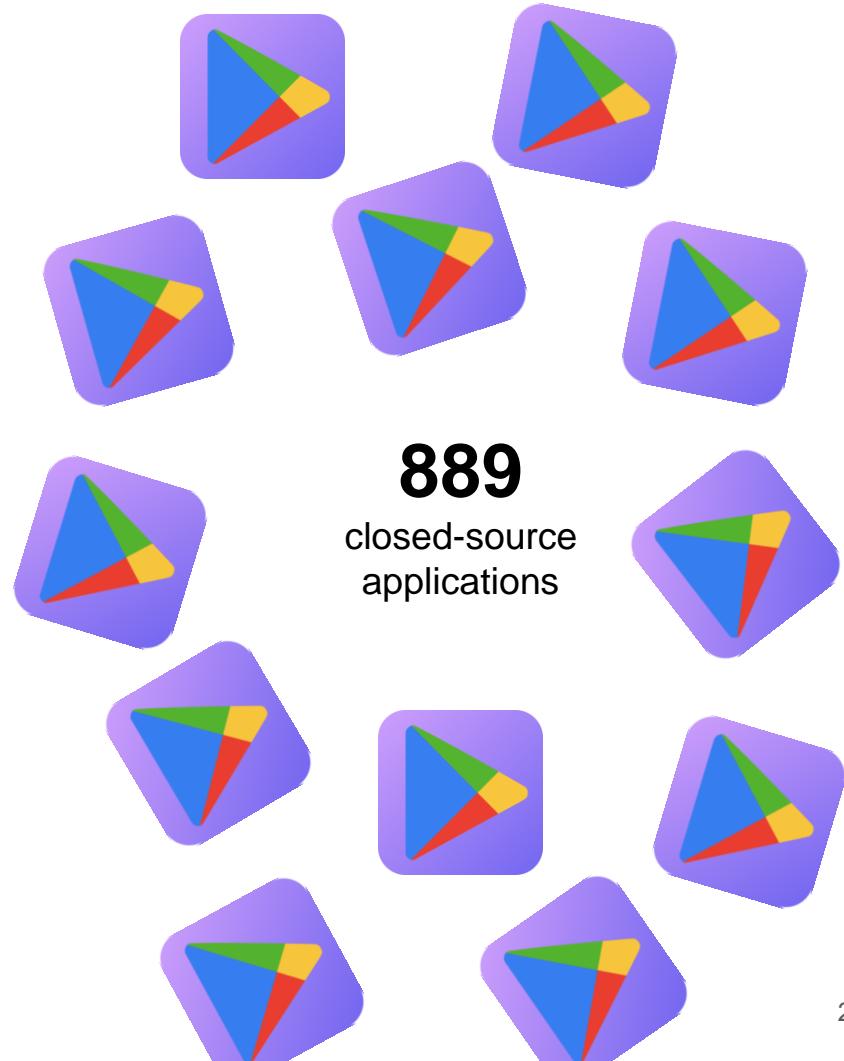
431

open-source
applications



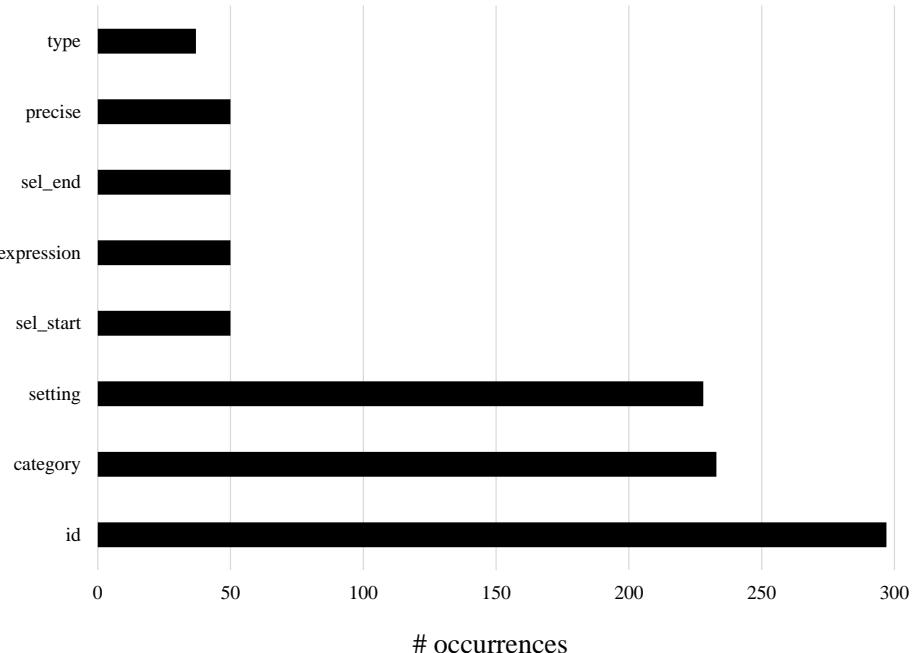
889

closed-source
applications

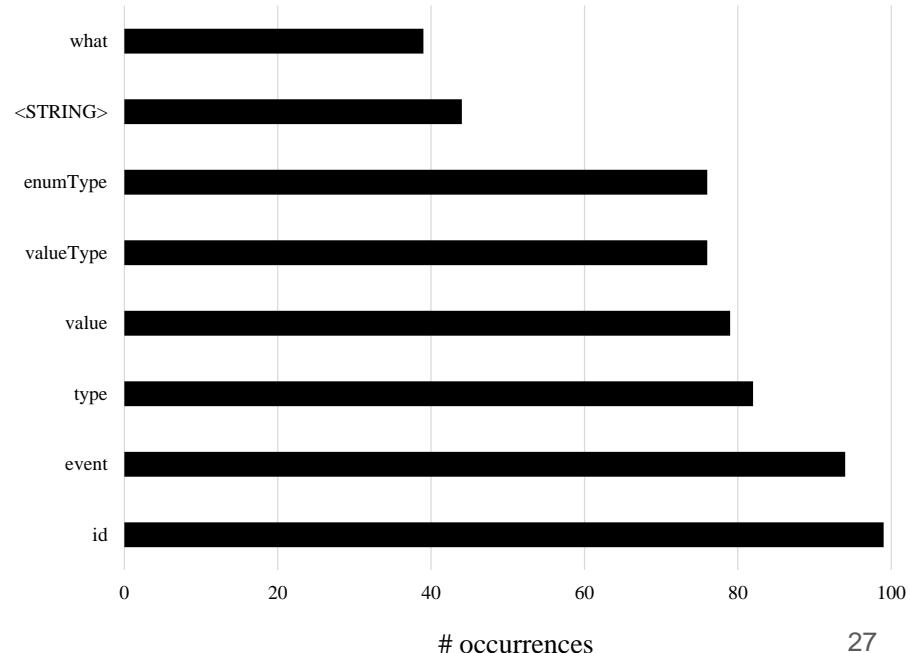


Top Query and JSON Keys

open-source projects

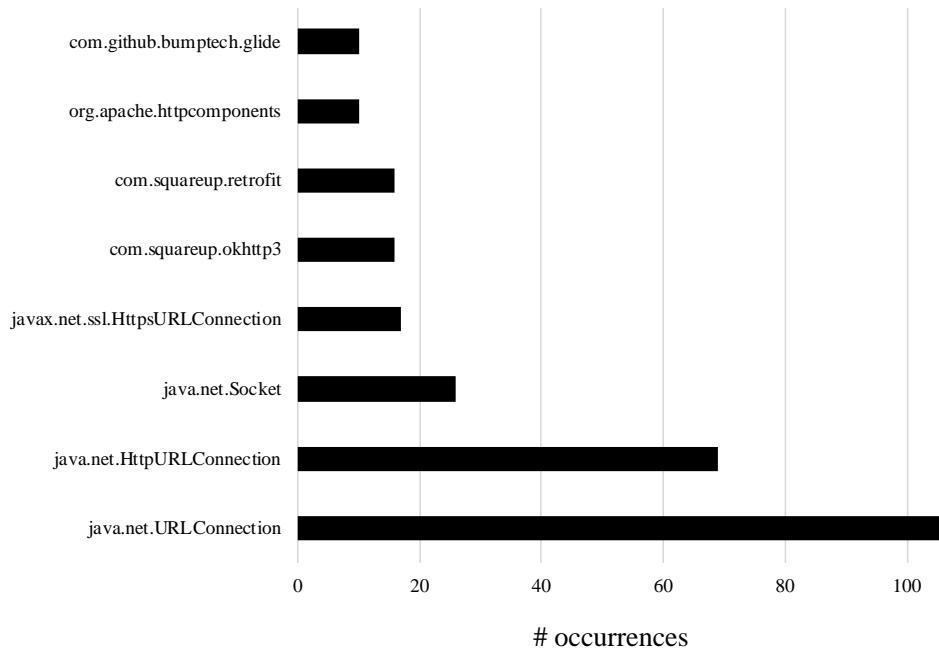


closed-source projects

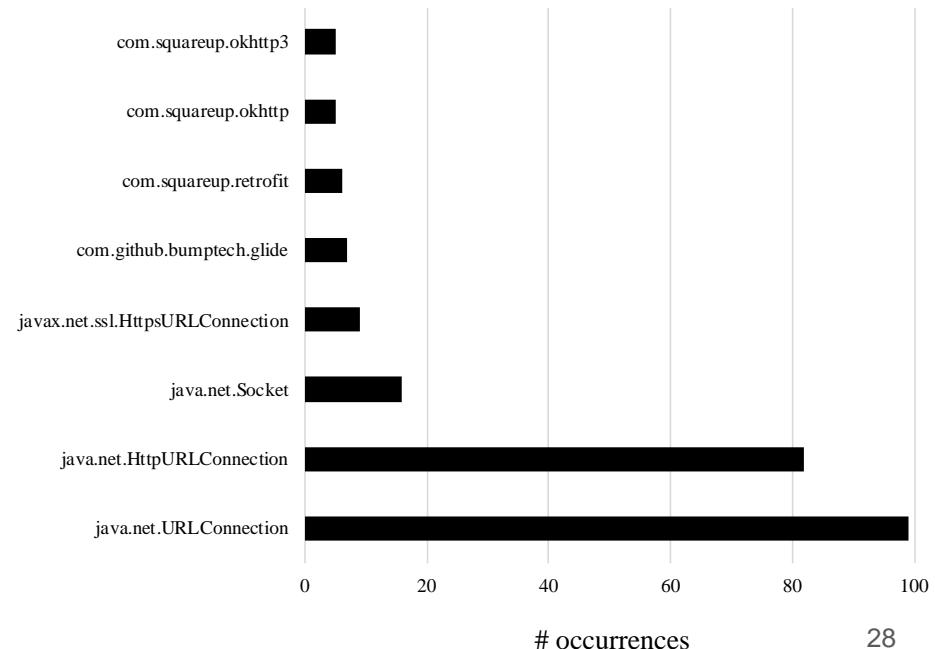


Library Usage

open-source projects

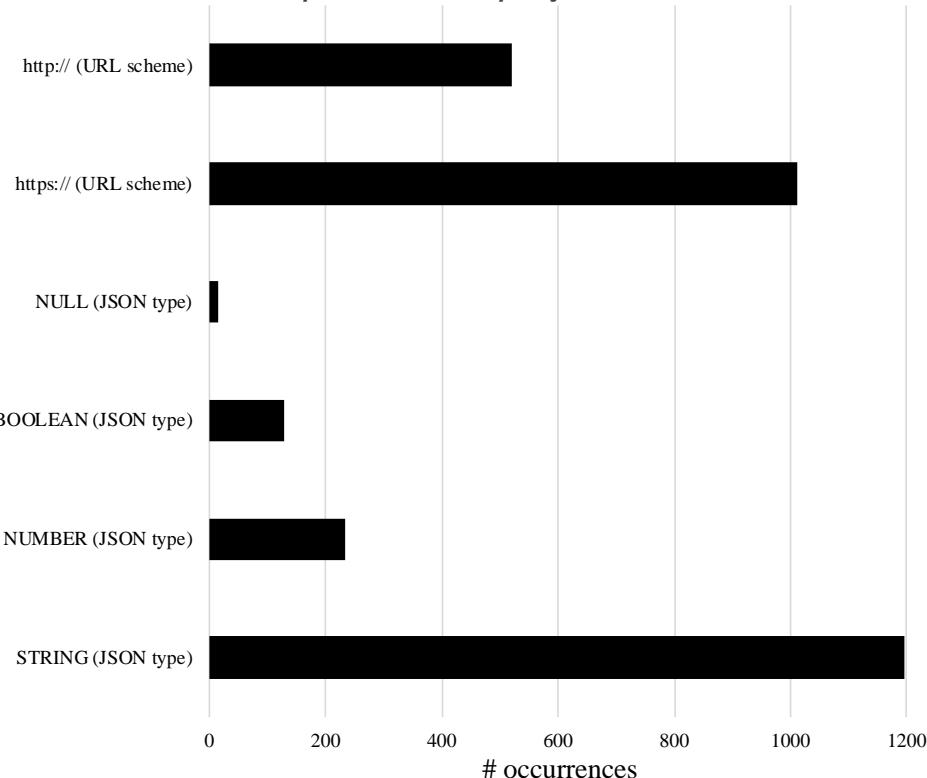


closed-source projects

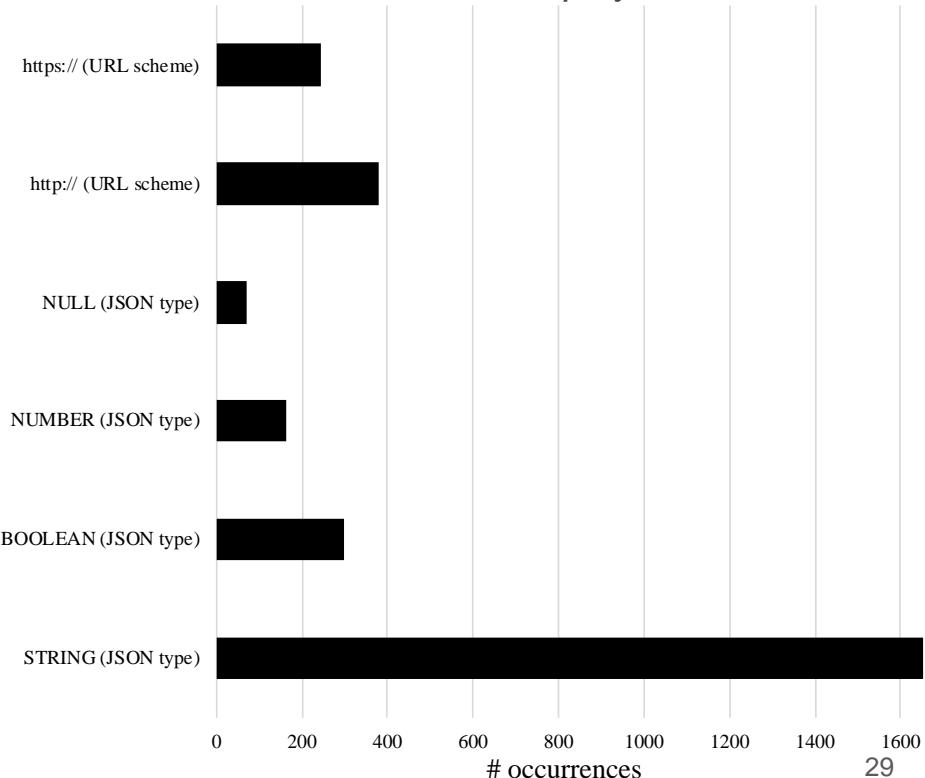


JSON Types and URL Schemes

open-source projects



closed-source projects



RQ4:

What configurations apply to
API endpoint servers found
in the wild?

Endpoint Server Analysis – What We Found

Outdated server software with known security vulnerabilities
(web server daemons, language parsers & interpreters)

Web views using HTTP protocol
(unprotected communication)

Disclosures of server configuration
(internal error message leaks, ...)

Hardcoded sensitive information
(API keys and other credentials)

Freely accessible APIs
(which are intended for private use)

Shell commands in request body (!)
(does not need any further explanations)

Jandrolyzer

Demo

Future Work

Jandrolyzer improvements

more {speed, library support, strategies}

More comprehensive static analyses

more applications, security-related investigations

More comprehensive dynamic analyses

accurate server assessments (OS, web server, programming languages)

+ server port testing (by using default passwords)

Conclusions

Decompilation with JADX (great tool, obfuscation, etc.)

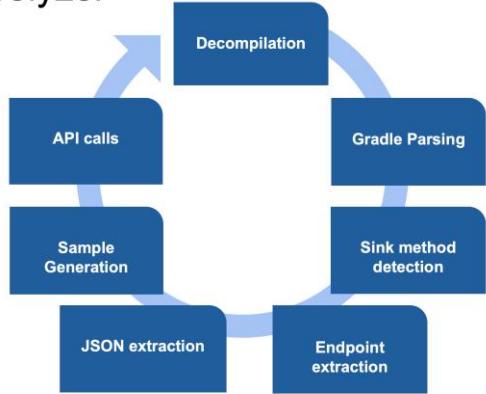
JavaParser (great tool, memory problems, symbol resolving bugs, etc.)

Open-source vs. closed-source apps (transmitted data, libraries, security, ads)

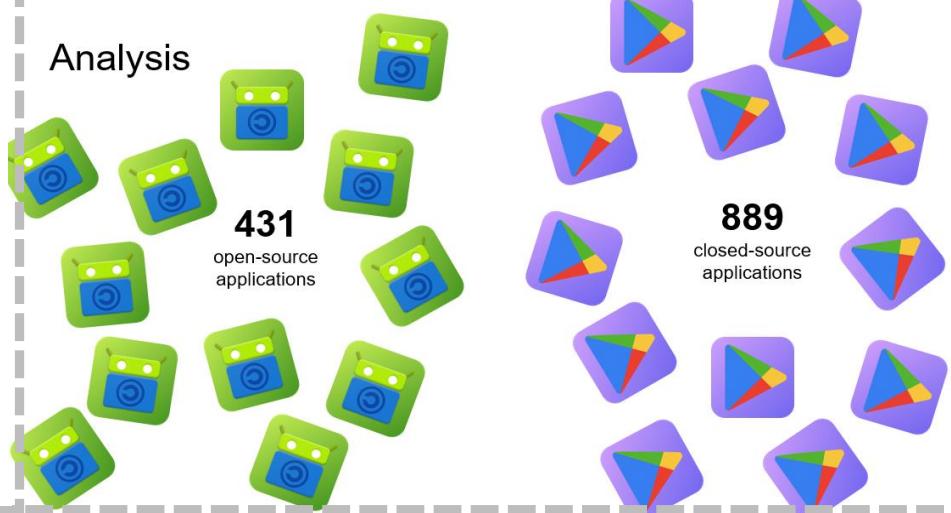
Outdated software in the wild

Implementation variations

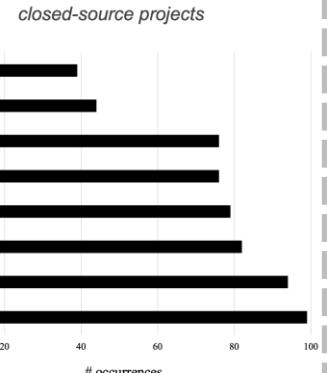
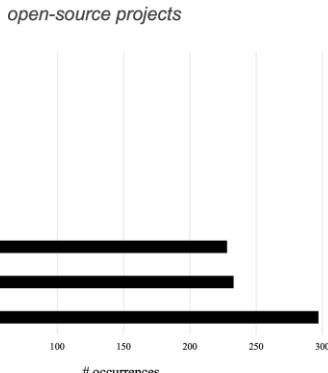
Jandrolyzer



Analysis



Top Query and JSON Keys



Endpoint Server Analysis – What We Found

Outdated server software with known security vulnerabilities
(web server daemons, language parsers & interpreters)

Web views using HTTP protocol
(unprotected communication)

Disclosures of server configuration
(internal error message leaks, ...)

Hardcoded sensitive information
(API keys and other credentials)

Freely accessible APIs
(which are intended for private use)

Shell commands in request body (!)
(does not need any further explanation)