Web Communication Analysis of Android Applications

Master Thesis - Final Presentation

Marc-Andrea Tarnutzer
09.04.2019
Web Communication

Request → http://api.snapchat.com/

Response

Request → https://www.admob.com/

Response
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 Programming Interface (API)

Request → API endpoint (server) → Response

Application server(s)
REpresentational State Transfer (REST)

- Client-Server Architecture
- Stateless
- Uniform interfaces
- Layered System
- Cacheable
- Code on demand
Data Schemes

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

```xml
<?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

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

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

Response:
```
HTTP/1.1 200 No Error
Content-Type: application/json; charset=UTF-8
Access-Control-Allow-Origin: *
Pragma: no-cache
Set-Cookie: PHPSESSID=u45n2me6f0jueflptpjmvj4d; 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
}
```
RQ1:
Which are the *prevalent web communication frameworks* used in mobile apps?
Found Web Communication Facilities
(manual analysis of 160 apps)

HTTP libraries
- Volley
- OkHttp
- Retrofit
- AndroidHttpClient
- AsyncHttpClient
- glide
- Ion

JSON libraries
- Gson
  github.com/google/gson
- org.json
  Included in Android
- Moshi
  github.com/square/moshi
REST Library Support

**without REST library**

```java
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.println("GET / HTTP/1.1\n
Host: www.google.ch\n
Connection: Close\n
\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.println((char) in);
}
bufferedReader.close();
```

**with REST library**

```java
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);
    }
});
```
JSON Library Support

without JSON library

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

with JSON library

```java
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();
}
```
Jandrolyzer

Introduction
Jandrolyzer

- Decompilation
- API calls
- Sample Generation
- JSON extraction
- Endpoint extraction
- Gradle Parsing
- Sink method detection
Analysis Step 1 / 7: Decompilation
Analysis Step 2 / 7: Gradle Parsing
Analysis Step 3 / 7: Sink method detection

com.koushikdutta.ion.builder.LoadBuilder

```java
Ion with(this)
    .load("http://someapi.com")
    .asJsonObject()
    .setCallback(new FutureCallback<JsonObject>() {
        @Override
        public void onCompleted(Exception e, JsonObejct result) {
            System.out.print("Result: " + result);
        }
    });
```
Analysis Step 4 / 7: Endpoint Extraction

**interface implementation**

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

RetrofitAPIEndpointInterface rApiInt = retrofit.create(RetrofitAPIEndpointInterface.class);
```

**interface declaration**

```java
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);
}
```

http://oldapi.com/api/loadNewsFromOldAPI
http://retrofiturl.com/api/loadUser/<INTEGER>
http://retrofiturl.com/api/loadUsers
http://retrofiturl.com/api/createUser
Analysis Step 5 / 7: JSON Extraction

```java
public void sampleReq() {
    String id = "AAAAABBBBBCCCC";
    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();
    }

    ...  
```
Analysis Step 6 / 7: Sample Generation

extracted URLs & JSON

https://somelservice.com/api/registerDevice?api_key=APIKEY

```json
{}
  "device_id": "AAAABBBCCCC",
  "api_key": "<STRING>"
}
```

extracted variables

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

Jaro-Winkler similarity: 0.8944...

final result

```json
{}
  "device_id": "AAAABBBCCCC",
  "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

- type
- precise
- sel_end
- expression
- sel_start
- setting
- category
- id

# occurrences

closed-source projects

- what
- <STRING>
- enumType
- valueType
- value
- type
- event
- id

# occurrences
Library Usage

open-source projects

- com.github.bumptech.glide
- org.apache.httpcomponents
- com.squareup.retrofit
- com.squareup.okhttp3
- javax.net.ssl.HttpsURLConnection
- java.net.Socket
- java.net.HttpURLConnection
- java.net.URLConnection

closed-source projects

- com.squareup.okhttp3
- com.squareup.okhttp
- com.squareup.retrofit
- com.github.bumptech.glide
- javax.net.ssl.HttpsURLConnection
- java.net.Socket
- java.net.HttpURLConnection
- java.net.URLConnection

# occurrences
JSON Types and URL Schemes

open-source projects

- http:// (URL scheme)
- https:// (URL scheme)
- NULL (JSON type)
- BOOLEAN (JSON type)
- NUMBER (JSON type)
- STRING (JSON type)

# occurrences

0 200 400 600 800 1000 1200

closed-source projects

- http:// (URL scheme)
- https:// (URL scheme)
- NULL (JSON type)
- NUMBER (JSON type)
- BOOLEAN (JSON type)
- STRING (JSON type)

# occurrences

0 200 400 600 800 1000 1200 1400 1600
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)