

All Your App Links are Belong to Us: Understanding the Threats of Instant Apps based Attacks

Yutian Tang¹, Yulei Sui², Haoyu Wang³, Xiapu Luo⁴, Hao Zhou⁴, Zhou Xu⁵.

1. ShanghaiTech University;
2. University of Technology Sydney;
3. Beijing University of Posts and Communications;
4. Hong Kong Polytechnic University;
5. Chongqing University;

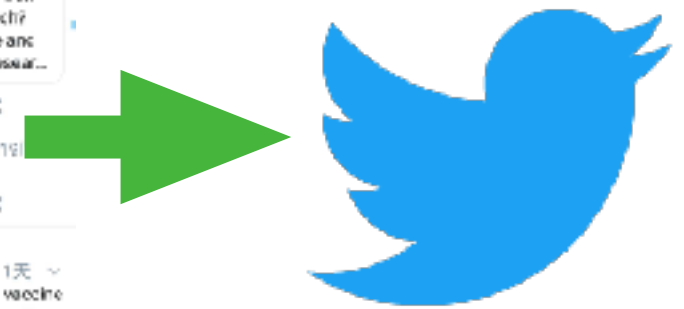
Web Browsing is Going Mobile

- Users spend more time on mobile devices [1]

- Mobile devices ~3.1 hours
- Laptops/Desktops ~2.2 hours



- Apps: the new web interface
 - Shorter loading time
 - Customized design
 - Million apps (Android + iOS)



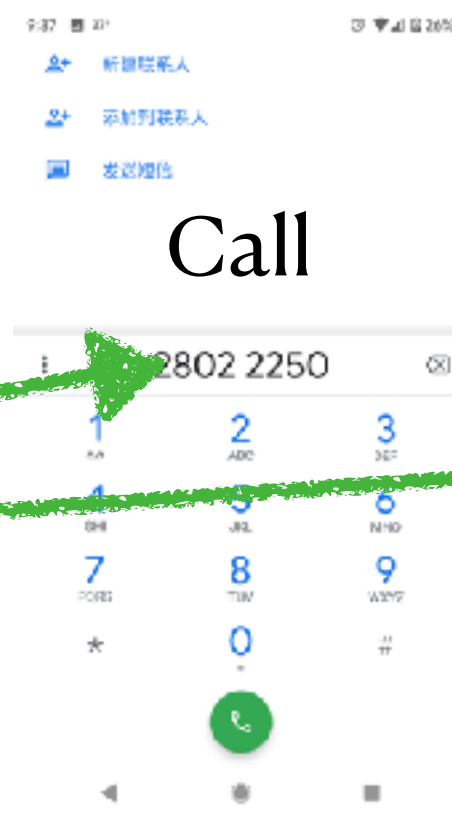
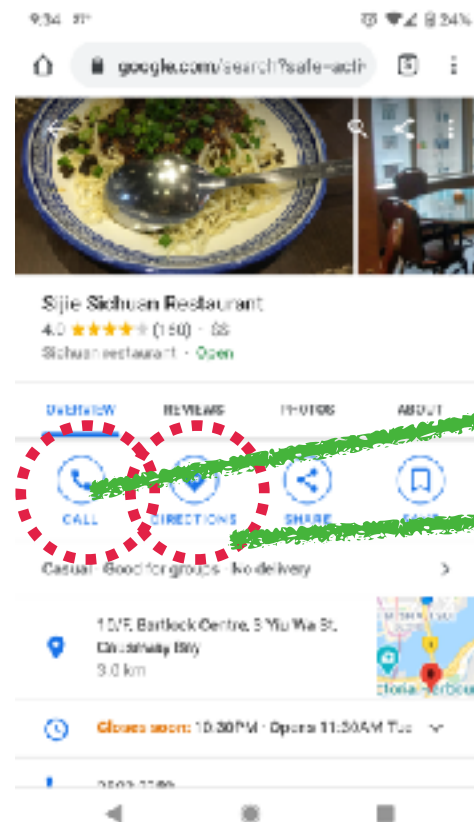
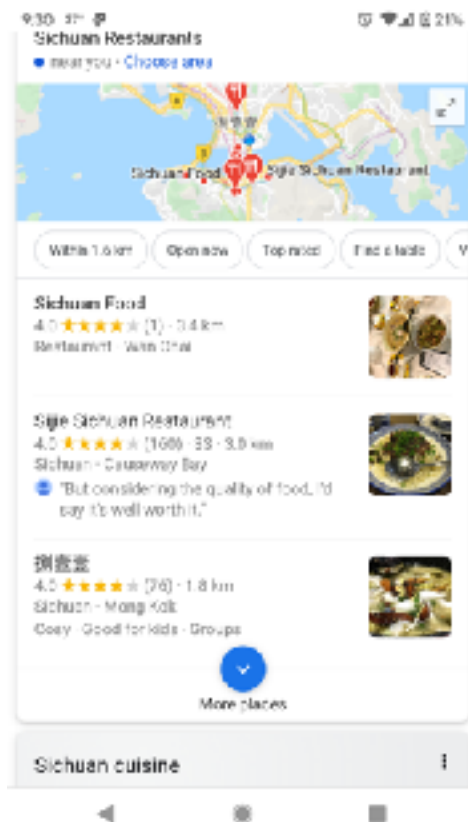
Apps vs. Mobile Websites

- Apps cannot fully replace websites yet!
 - Apps sit in a “walled garden”;
 - Difficult to navigate across apps;
 - Difficult to search and access in-app content globally
- Apps + mobile websites eco-system
 - Complementary to each other
 - Likely to co-exist (for a long time)

Web-App Communication via Deep Links

Deep Links

- Deeper integration of websites and apps;
- Mobile deep links: URIs pointing to pages inside apps



Call

Google Map

Greatly improve the experience!

Hijacking Risks of Deep Links

- Scheme URL: mobile deep link v1.0
 - Designed for functionality, **no security feature**;
 - Apps can define and register their own schemes to the OS;

fb://share/?data=1&sessionID=123



Manifest.xml

intent filters

fb://share/*

- [Mobisys'11] [CCS'14][CCS'15]

Any app can register this scheme in Intent Filter (Phishing attacks)

App Link: Deep Link v2.0

Prevents Link Hijacking

- Android 6.0
- App Link
 - HTTP/HTTPS links only; no custom schemes;
 - Requires **app link association** (domain side verification)
 - fb:// —> <https://facebook.com/>

Is it secure?

App Link and Instant App

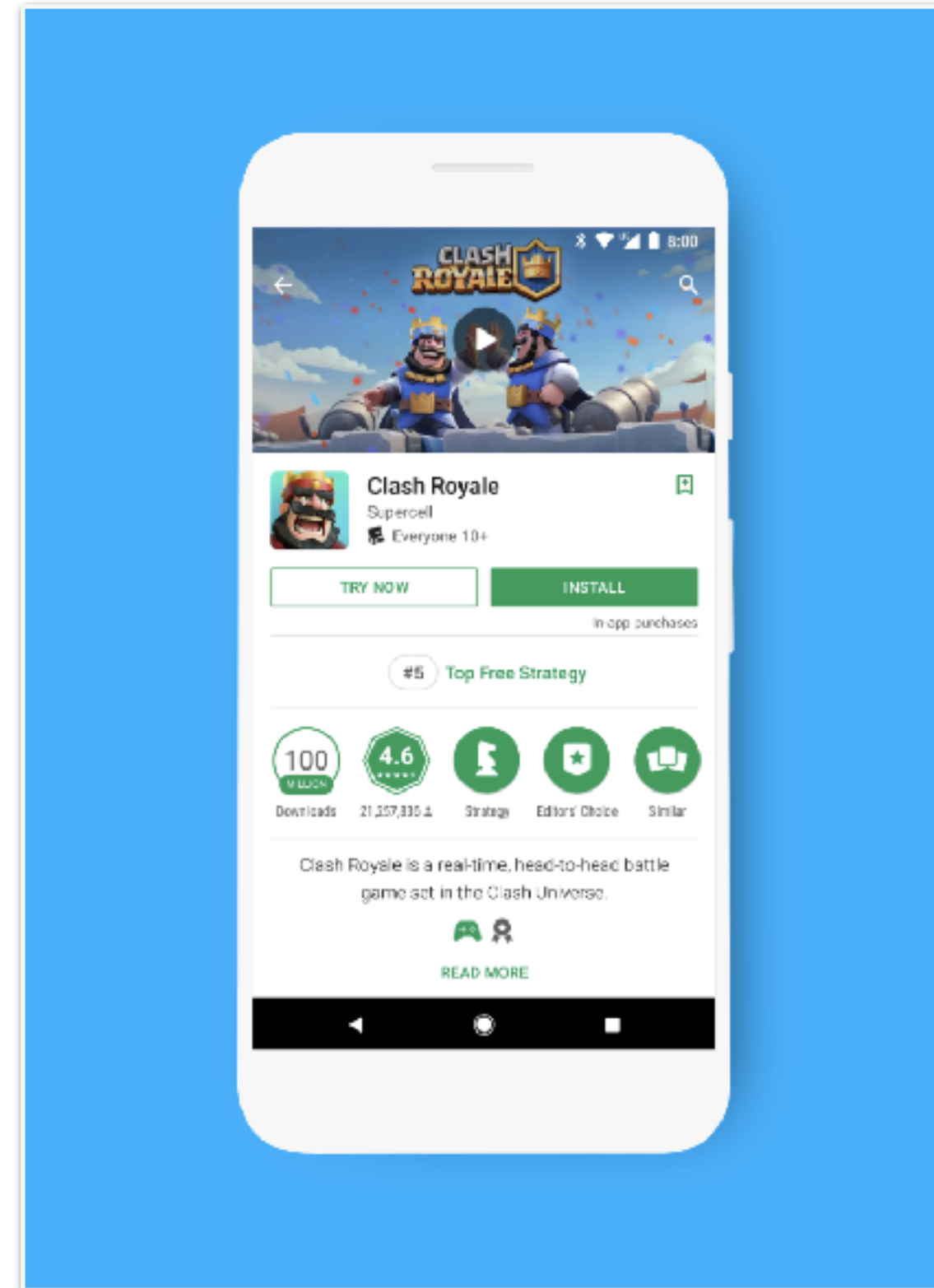
Is it secure?

NO! Instant App based Attacks!

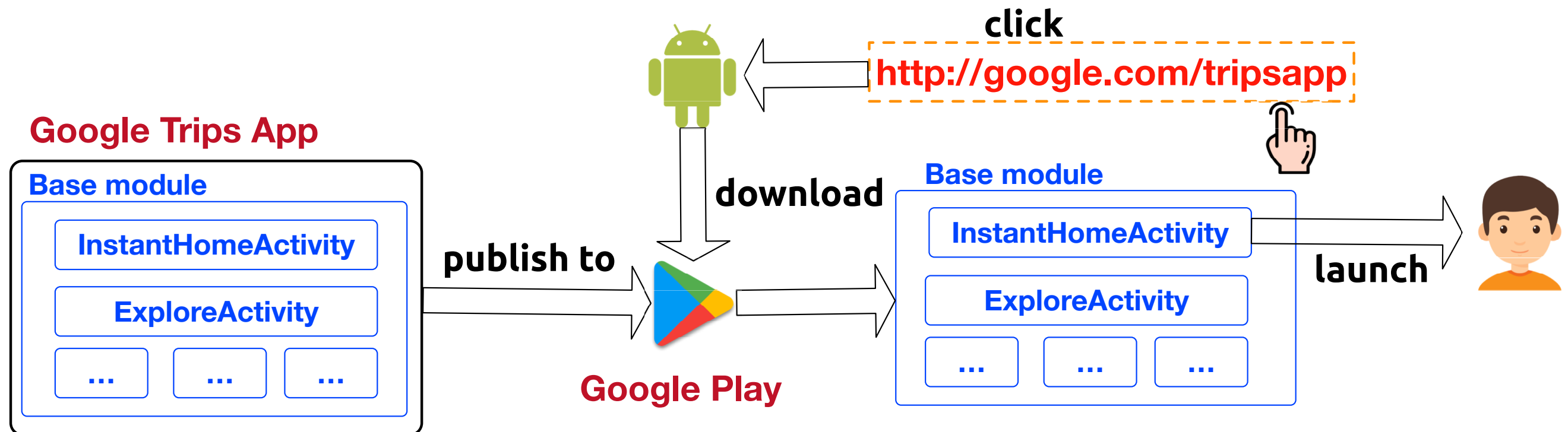


Confirmed by Google Security Team

- Instant App: People can use an app or game **without installing it first**.
- Increase engagement with apps or gain more installs by surfacing your instant app across the Play Store



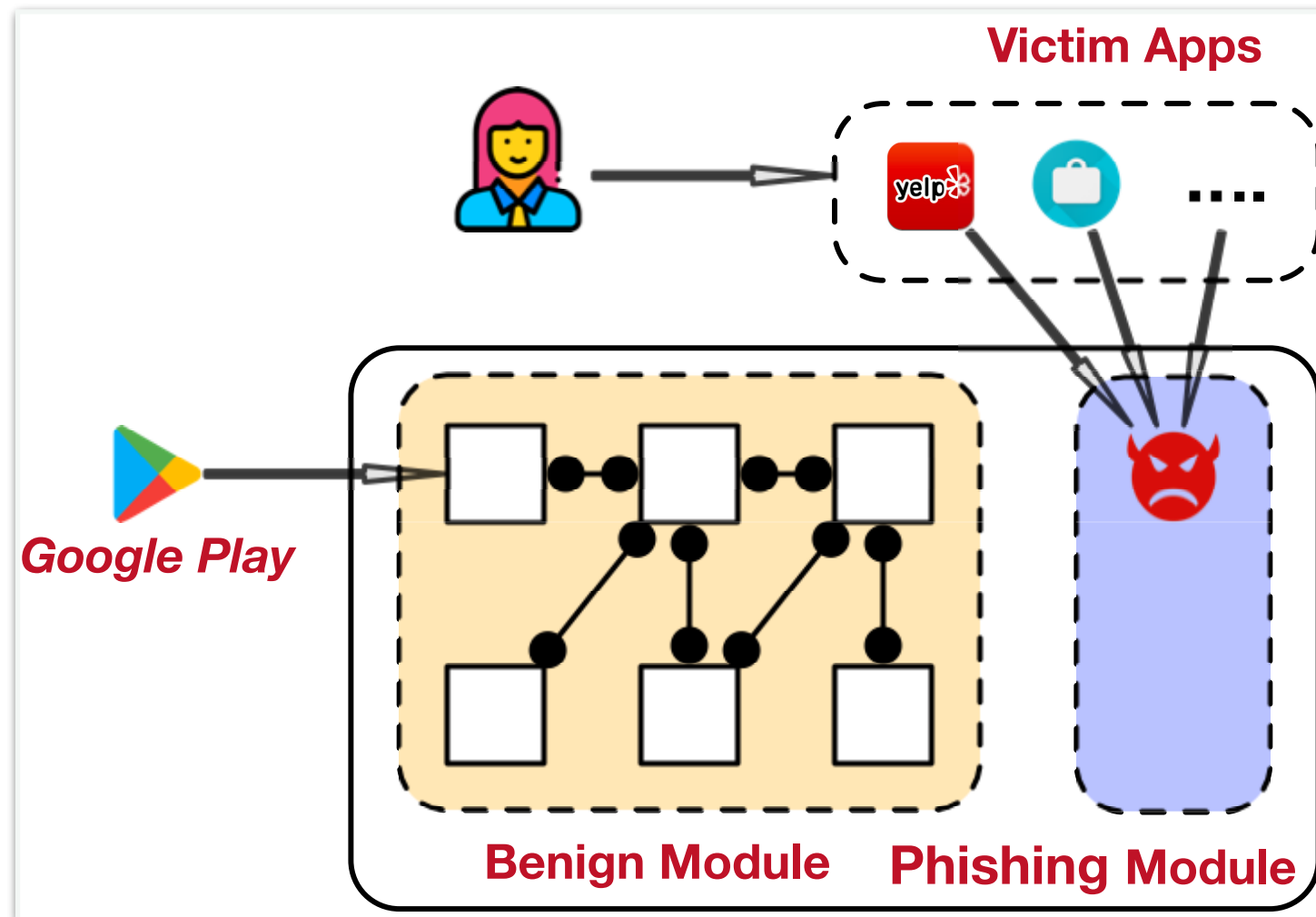
Android Instant App Workflow



- Instant App
 - Module-based organization;
 - Click an URL —> module is downloaded and launched;

(<http://google.com/tripsapp> —> Google Trips Apps)

The Architecture of Malicious Instant App

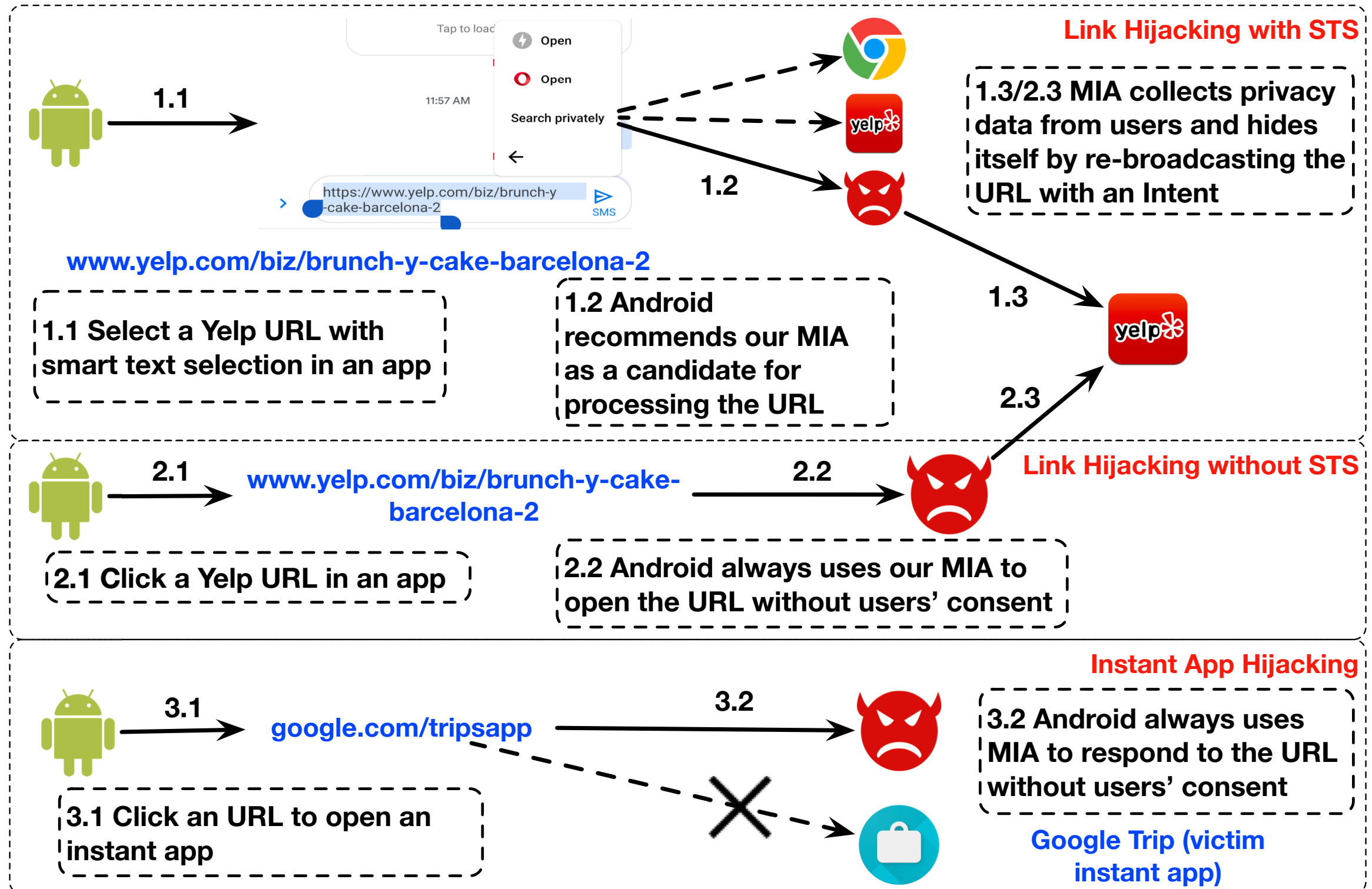


Install from the official channel (Google Play); [We already bypass Google Play's security checking]

Phishing Module ==> simulate login page;

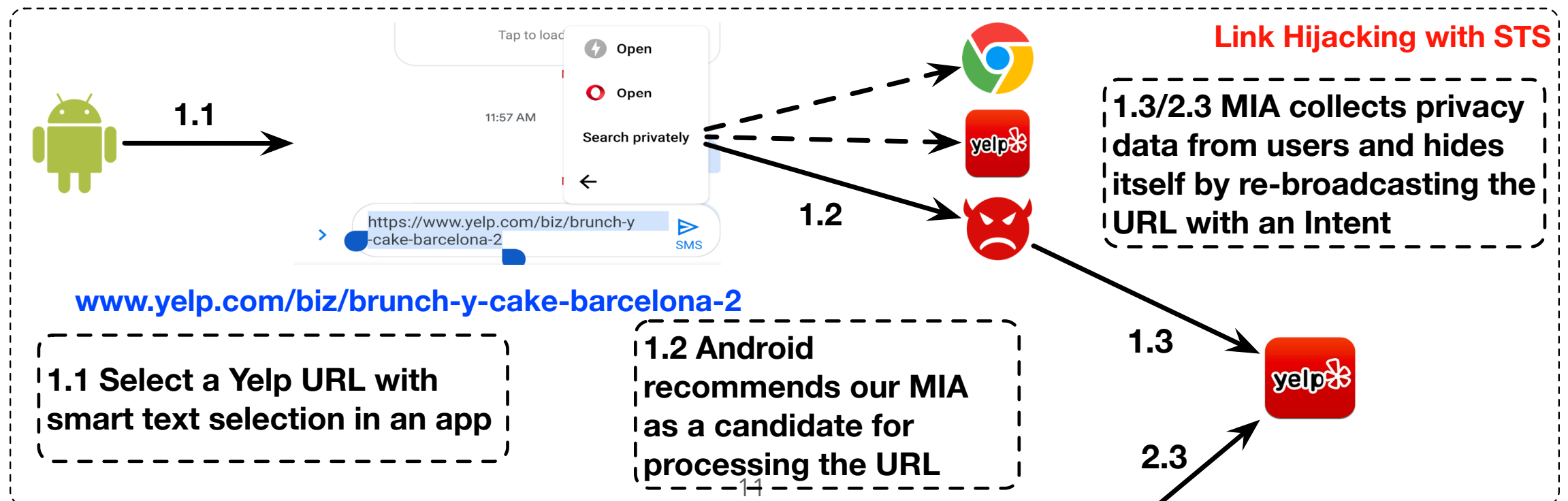
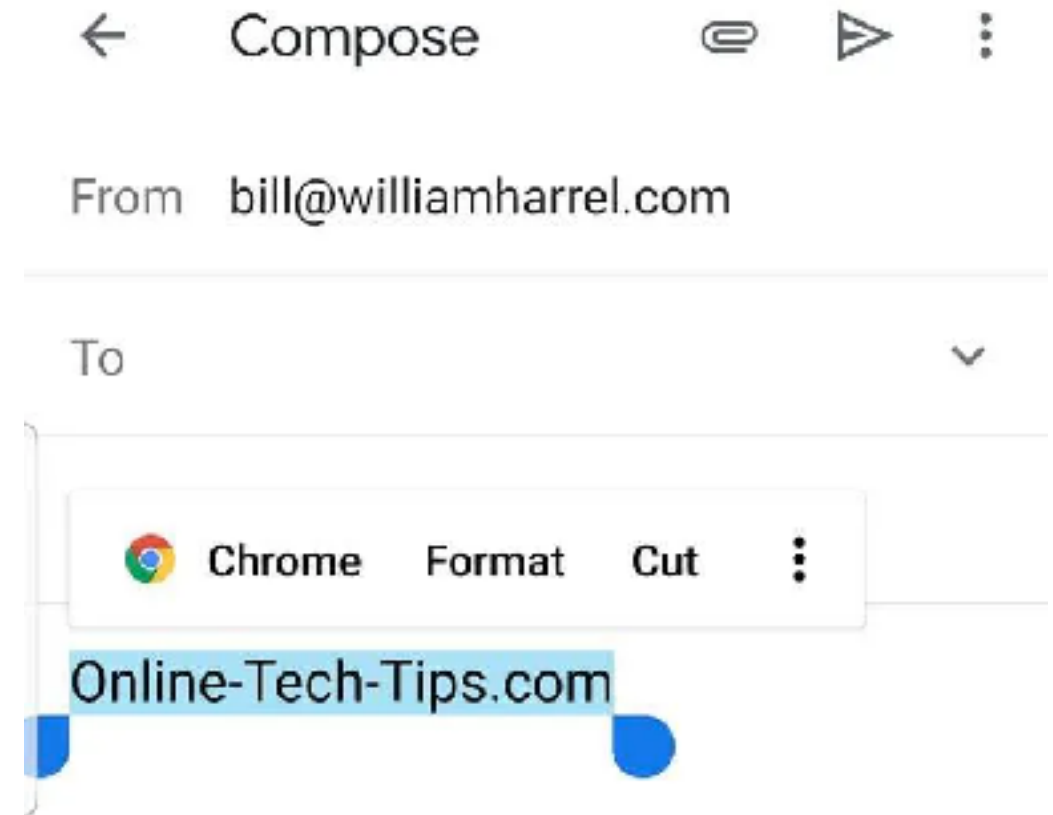
(a clone page for phishing);

Attacks



Attack 1: Link Hijacking with STS

- STS (Smart text selection)
 - Android 8.0 +;
 - Text: web, phone, mail, map ...;
- **Victim:** app;
- **Attack Vector:** our malicious instant app (MIA)



Attack 1: Link Hijacking with STS (2)

```
<activity android:name=".MainActivity">
    ...
    <meta-data
        android:name="default-url"
        android:value="https://www.example.org/main" />
    <intent-filter android:autoVerify="true">
        <action android:name="android.intent.action.VIEW" />
        <category
            android:name="android.intent.category.DEFAULT" />
        <category
            android:name="android.intent.category.BROWSABLE" />
        <data android:host="www.<my-own-site>.org"
            android:pathPattern="/main"
            android:scheme="http" />
    </intent-filter>
</activity>
<activity android:name=".LoginActivity">
    <intent-filter>
        <action android:name="android.intent.action.VIEW" />
        <category
            android:name="android.intent.category.DEFAULT" />
        <data android:host="www.yelp.com"
            android:pathPattern="/.*" android:scheme="http" />
        <data android:host="www.yelp.com"
            android:pathPattern="/.*" android:scheme="https" />
    </intent-filter>
</activity>
```

App Link

Default/Install

for installing &
launching the
instant app

Deep Link

Hijacking

for hijacking

www.yelp.com/biz/brunch-y-cake-barcelona-2

Attack 1: Link Hijacking with STS (3)

```
private static List<LabeledIntent> createForUrl(Context context, String
text) {
    if (Uri.parse(text).getScheme() == null) {
        text = "http://" + text;
    }
    return Arrays.asList(new
LabeledIntent(context.getString(com.android.internal.R.string.browse),
context.getString(com.android.internal.R.string.browse_desc), new
Intent(Intent.ACTION_VIEW,
Uri.parse(text)).putExtra(Browser.EXTRA_APPLICATION_ID,
context.getPackageName()), LabeledIntent.DEFAULT_REQUEST_CODE));
}
```

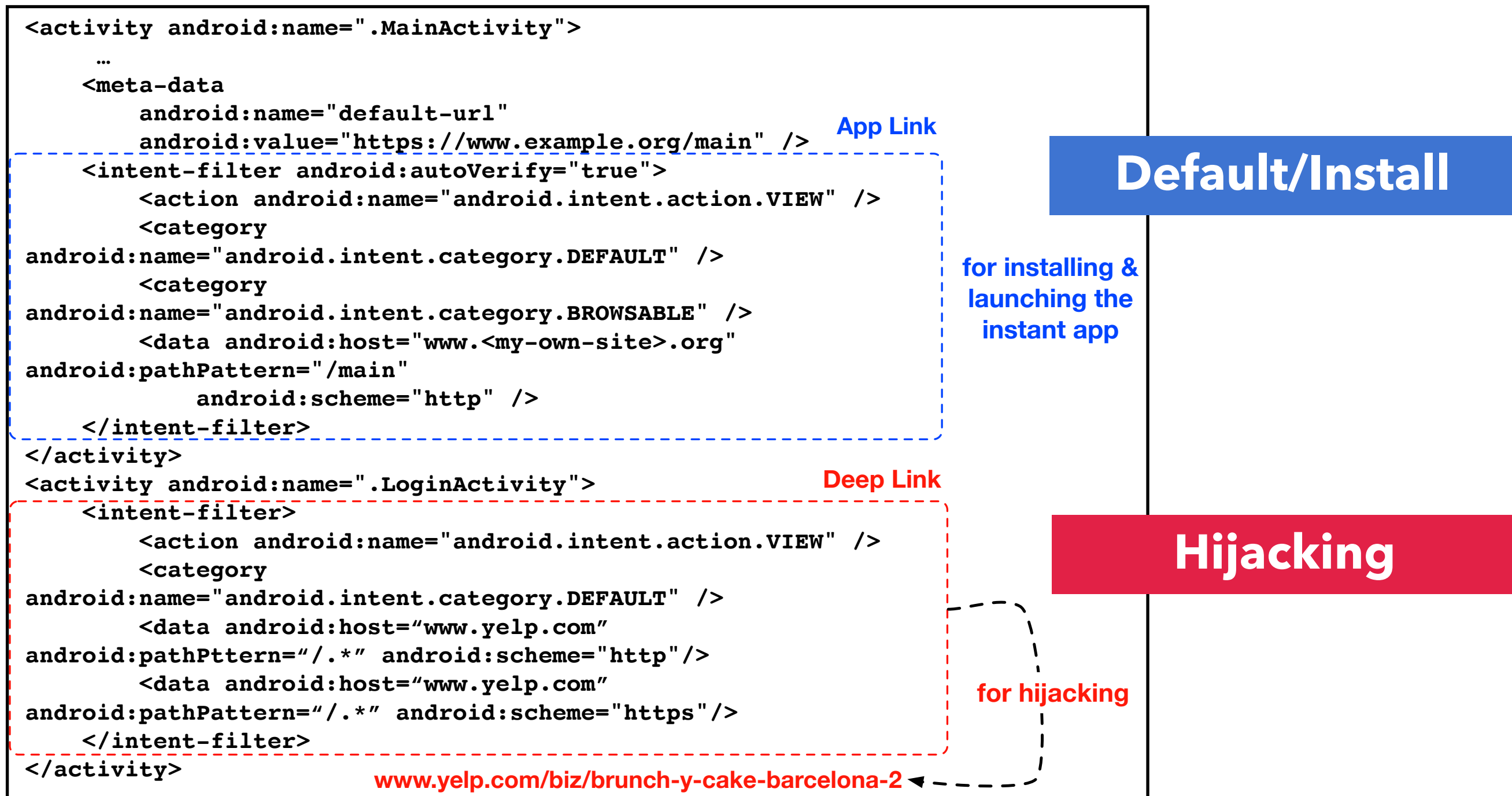
- If an URL is selected, Android looks up for all apps that can respond to the URL.
- Then, smart text selection (STS) suggests all these apps for users to select.
- Thus, STS can suggest our MIA to users.

Attack 2: Link Hijacking without STS

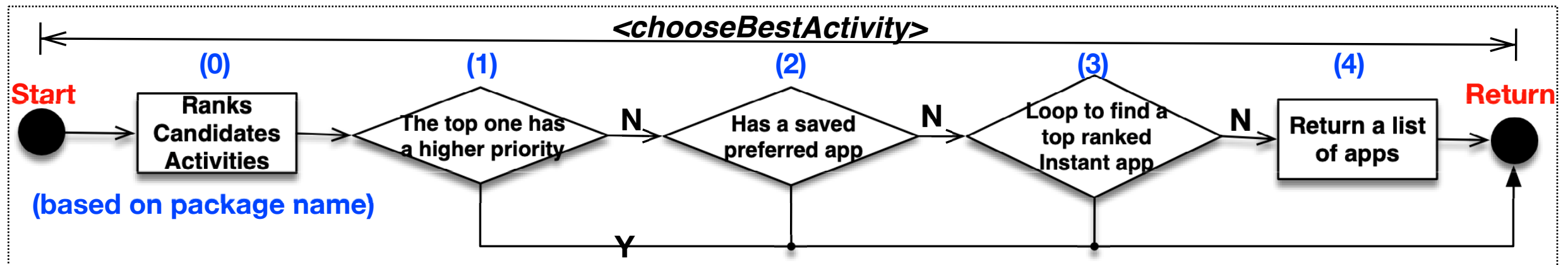


- **Victim:** app;
- **Attack Vector:** our malicious instant app (MIA)
- 2.1 Click a Yelp URL in an app (e.g., Message)
- 2.2 Android **always uses** our malicious instant app (MIA) to open the URL;

Attack 2: Link Hijacking without STS (2)

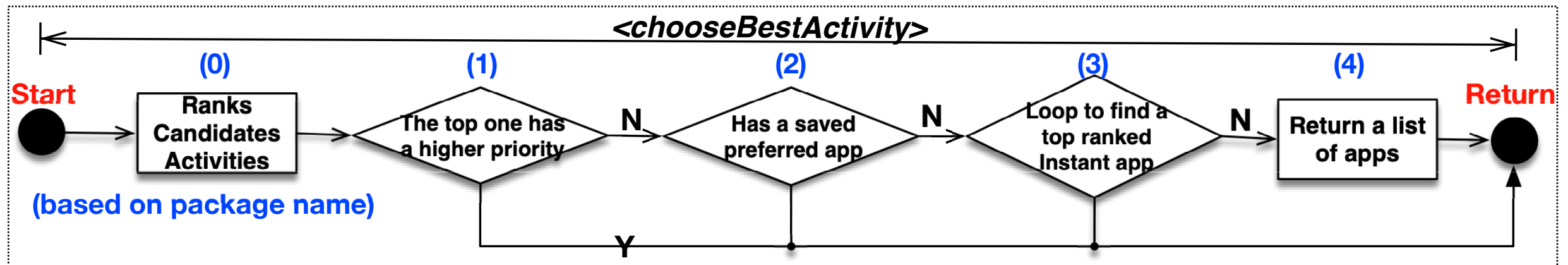


Attack 2: Link Hijacking without STS (3)




- Malicious instant app (MIA) installed from Google Play;
- Victim app & malicious instant app;
- Android ==> `chooseBestActivity ()`;

Attack 2: Link Hijacking without STS (4)

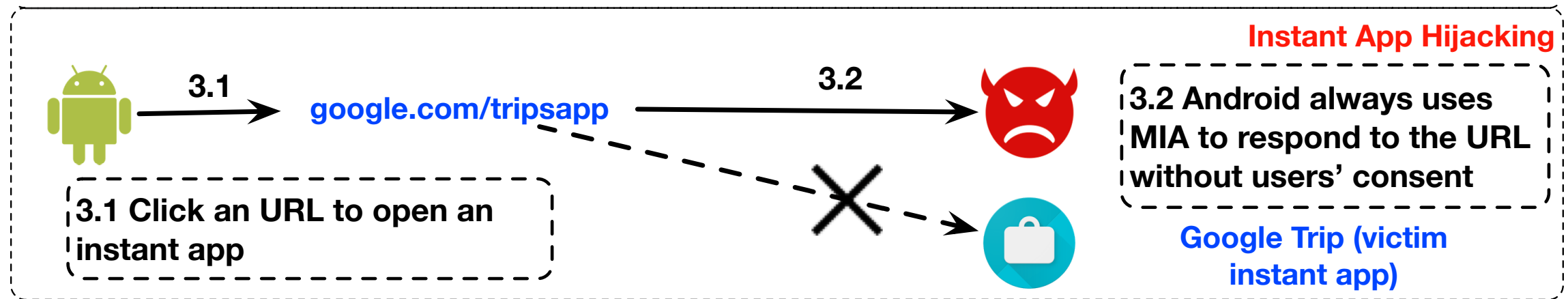


chooseBestActivity()

1. Ranks candidates Activity (based on package name);
2. Check whether one owns a higher priority?
3. Check whether there is a saved preferred app?
4. Instant App; ←  **Instant App > normal app**
5. Return a list of apps (use side)

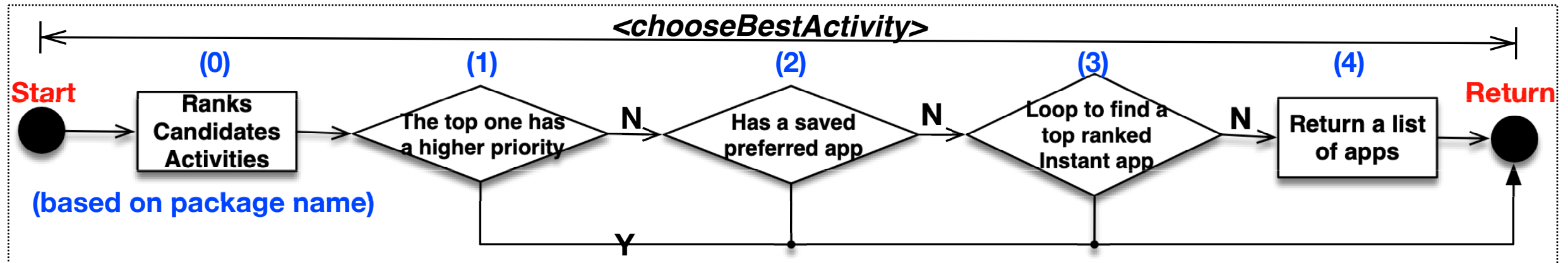


Attack 3: Instant App Hijacking



- **Victim:** Instant App;
- **Attack Vector:** our malicious instant app (MIA);
- 3.1 Click an URL to open an instant app;
- 3.2 Android **always uses** our MIA to respond to the URL rather than the victim instant app (Google trip in this case)

Attack 3: Instant App Hijacking (2)



chooseBestActivity()

1. Ranks candidates Activity (based on package name);
2. Check whether one owns a higher priority?
3. Check whether there is a saved preferred app?
4. Instant App; ← **Instant App > normal app**
5. Return a list of apps (use side)

2 or more Instant Apps?

(1) Ranks based on package name

a.b.XXX < com.google.android.apps.travel.onthego

Vulnerability

```
<activity android:name=".MainActivity">
  ...
  <meta-data
    android:name="default-url"
    android:value="https://www.example.org/main" />
  <intent-filter android:autoVerify="true">
    <action android:name="android.intent.action.VIEW" />
    <category
      android:name="android.intent.category.DEFAULT" />
    <category
      android:name="android.intent.category.BROWSABLE" />
    <data android:host="www.<my-own-site>.org"
      android:pathPattern="/main"
      android:scheme="http" />
  </intent-filter>
</activity>
<activity android:name=".LoginActivity">
  <intent-filter>
    <action android:name="android.intent.action.VIEW" />
    <category
      android:name="android.intent.category.DEFAULT" />
    <data android:host="www.yelp.com"
      android:pathPattern="/.*" android:scheme="http" />
    <data android:host="www.yelp.com"
      android:pathPattern="/.*" android:scheme="https" />
  </intent-filter>
</activity>
```

App Link

for installing & launching the instant app

Deep Link

for hijacking

www.yelp.com/biz/brunch-y-cake-barcelona-2

Default/Install

Hijacking

- Instant App > Normal App.
- Multiple Instant App ==> Package ranking.

Detecting Tool: VDetector

- FlowDroid
- Program Dependence Graph (PDG) \rightarrow UI-oriented Program Dependency Graph (UPDG)
- $n = \{uid, utype, a, c, o\}$
 - uid: id of an UI element;
 - utype: type of an UI element; (e.g., label)
 - a: Activity context;
 - c: callback method;
 - o: original PDG node;

RQ1. Are real-world apps correctly configure app links?

- 200,000 Google Play apps + 200,000 Tencent-Myapp apps;
- 18.0% Google Play apps correctly configure the app links;
- 3.1% Tencent-Myapp apps correctly configure the app links;
- Common errors:
 - Incorrect JSON formatting (not a valid JSON file);
 - Incorrect fields (undefined fields, some required/mandatory fields are missing; typo);
 - Incorrect namespace (the namespace must be “android_app” or “web”);

RQ2. Are real-world apps robust to the link hijacking attack with STS?

- 200,000 Google Play apps + 200,000 Tencent-Myapp apps;
- 53,619 Google Play apps (26.8%) that are vulnerable to link hijacking attacks with STS attack;
- 54,650 Tencent-Myapp apps (27.3%) that are vulnerable to link hijacking attacks with STS attacks;

RQ3. Are real-world apps robust to the link hijacking attack without STS?

- 200,000 Google Play apps + 200,000 Tencent-Myapp apps;
- 57,442 Google Play apps (28.7%) that are vulnerable to link hijacking attacks without STS attack;
- 62,496 Tencent-Myapp apps (31.2%) that are vulnerable to link hijacking attacks without STS attacks;

RQ4. Are Instant App robust to Instant App Hijacking?

- 36 real-world instant apps out of 200,000 Google Play apps
- All instant apps are vulnerable to instant app hijacking attack.

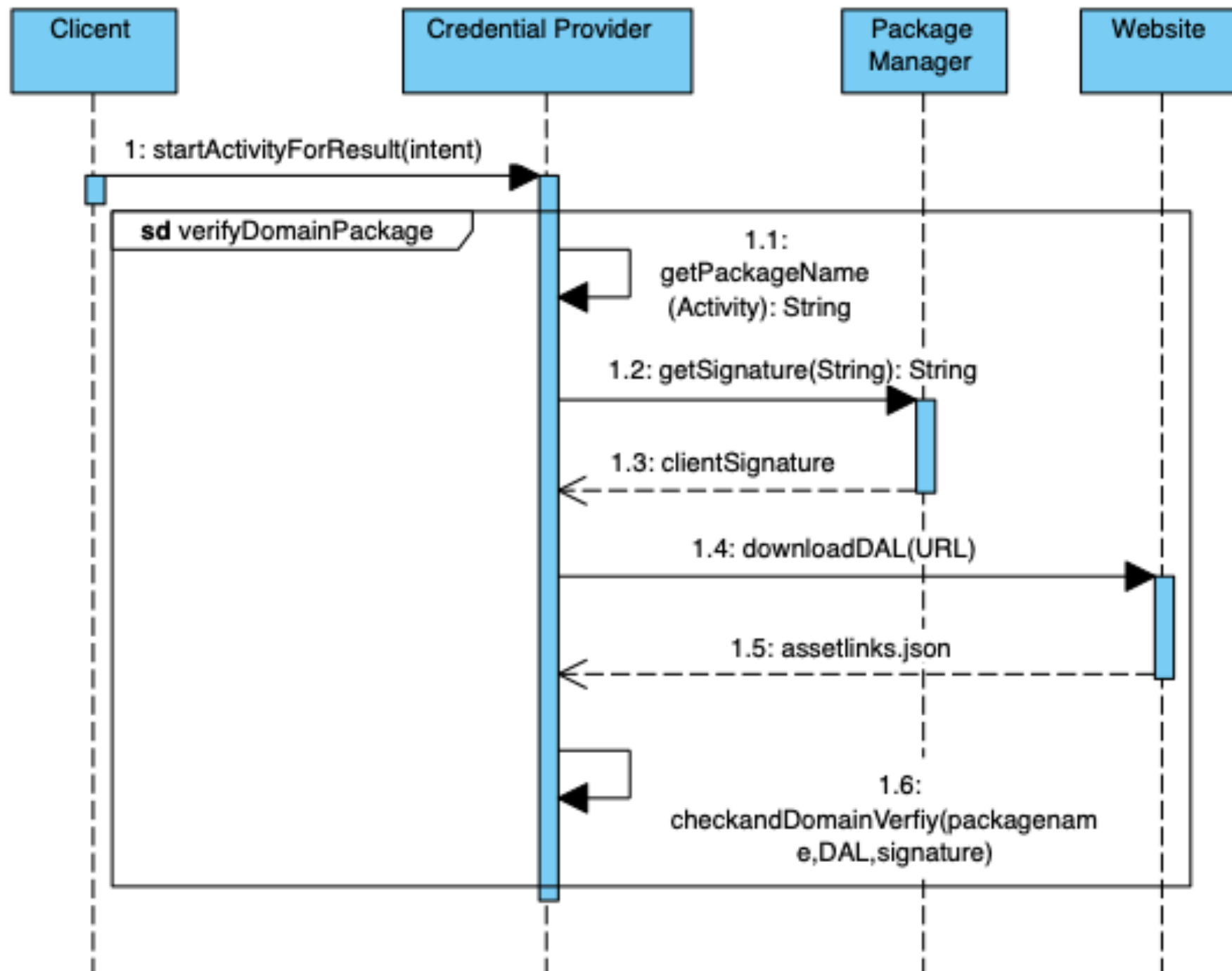
RQ5. Accuracy?

- 800 apps from Google Play;
- Manually check whether they can be attacked;
- Our tool reports 276 vulnerable apps and all can be exploited;
- 100% precision, 87% recall;
- Missing following apps:
 - Native code; (c/c++ code for UI management);
 - Third-party UI frameworks;

Countermeasure

- Solution 1: If a developer plans to use `TextView` (i.e. label) in an app, (s)he can use the `setTextSelectable(false)` to prevent any text selection in a label;
- Solution 2: If a developer plans to use `EditText` (editable text UI element), (s)he can use the `setMovementMethod(null)` to prevent any text selection in a label;
- Solution 3: Developers can override the `isSuggestionsEnabled()` in default `TextView/EditText` to disable smart text selection (STS);
- Solution 4: `setCustomSelectionActionModeCallback` API allows developers to customize the popup menu if a piece of text is selected.

Countermeasure (2)



`verifyDomainPackage()`

Thanks

If you have any questions, please feel free to contact Yutian Tang: tangyt1@shanghaitech.edu.cn