Apposcopy: Semantics-Based Detection of Android Malware through Static Analysis

Yu Feng, Saswat Anand, Isil Dillig and Alex Aiken

FSE 2014

Presented by ke tian

Outlines:

- What is the problem?

  identify Android malware

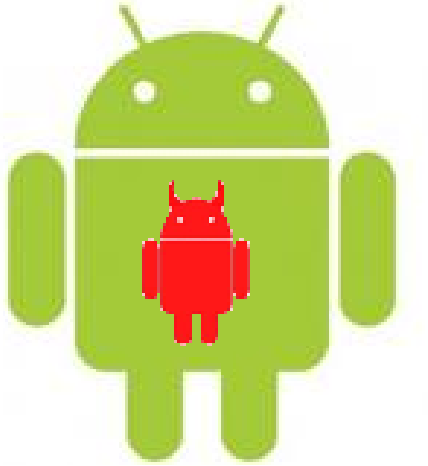
- What is the solution/contribution?

  signature based specification with graph assistant(ICCG)

- How efficient is the solution?

  low false positive+ false negative

Android attacks

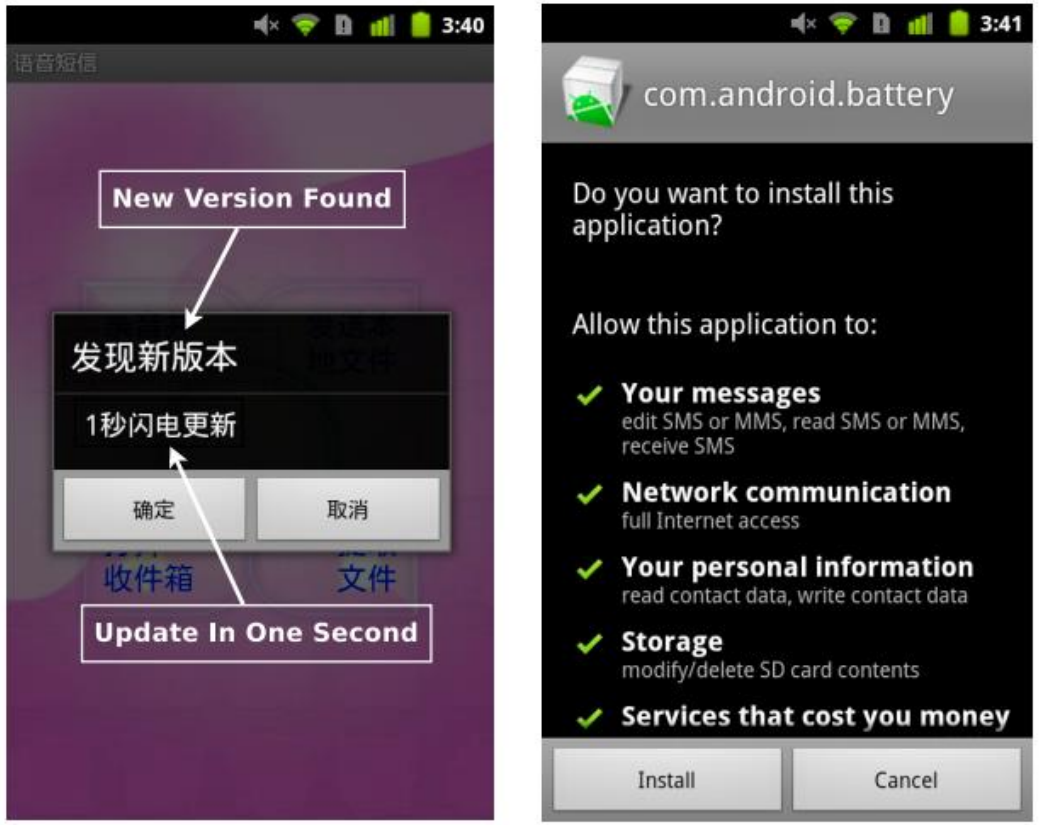


Repackaging

86.0% of # apps in Genome

Trojan

Stealing/sniffing



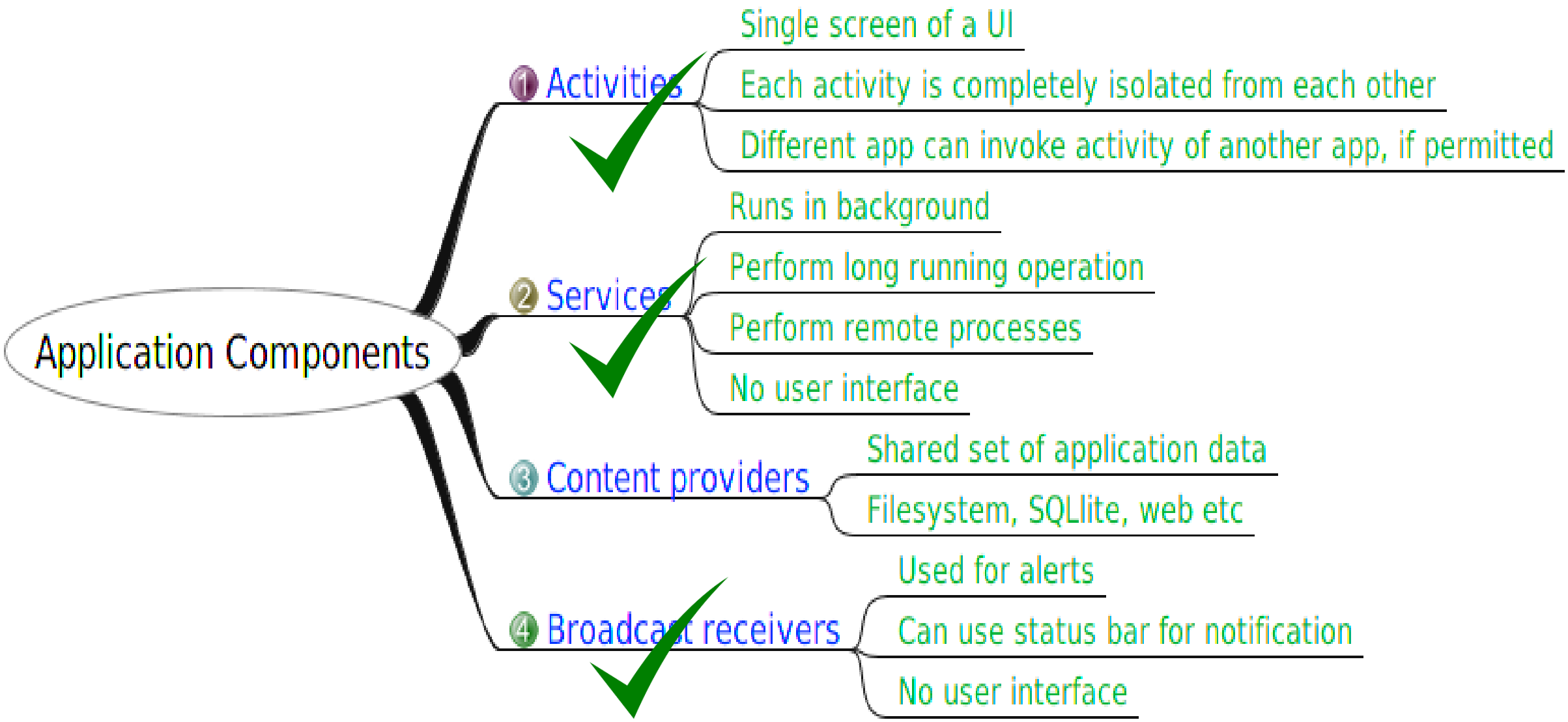
(a) The Update Dialogue (b) Installation of A New Version

Figure 2. An Update Attack from BaseBridge

Update attack

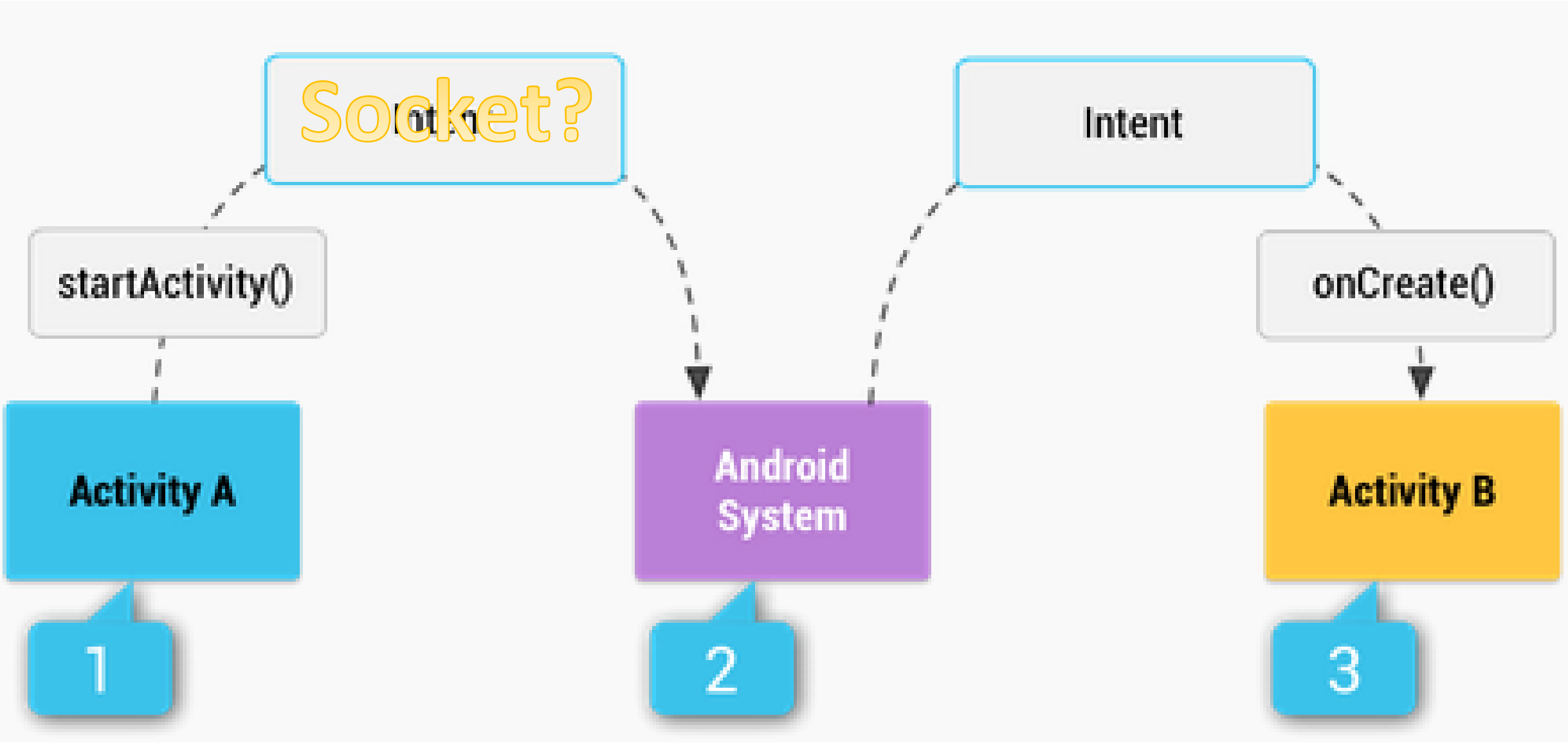
[1]Dissecting Android Malware: Characterization and Evolution

Android Components

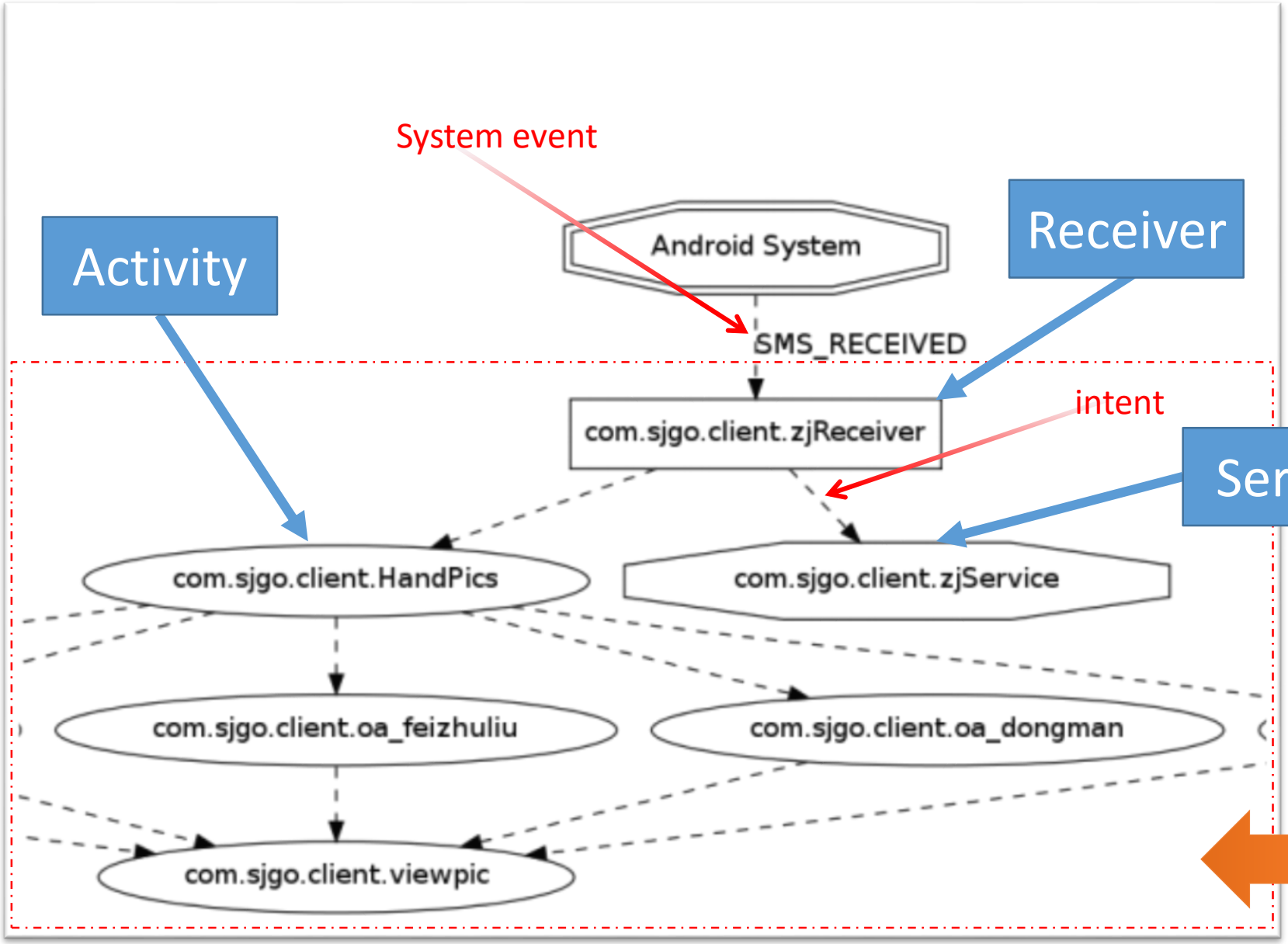


http://mithileshjoshi.blogspot.com/2015/06/what-is-android-application-components.html

Communications in Android



ICCG (inter-component call graph)



Node: component name

Edge: 1)component A
--(start)-- >
component B
2)system events

Signature

```
1.  GDEvent(SMS_RECEIVED).
2.  GDEvent(NEW_OUTGOING_CALL).
3.  GoldDream :-  receiver(r),
4.                icc(SYSTEM, r, e, _), GDEvent(e),
5.                service(s), icc*(r, s),
6.                flow(s, DeviceId, s, Internet),
7.                flow(s, SubscriberId, s, Internet).
```

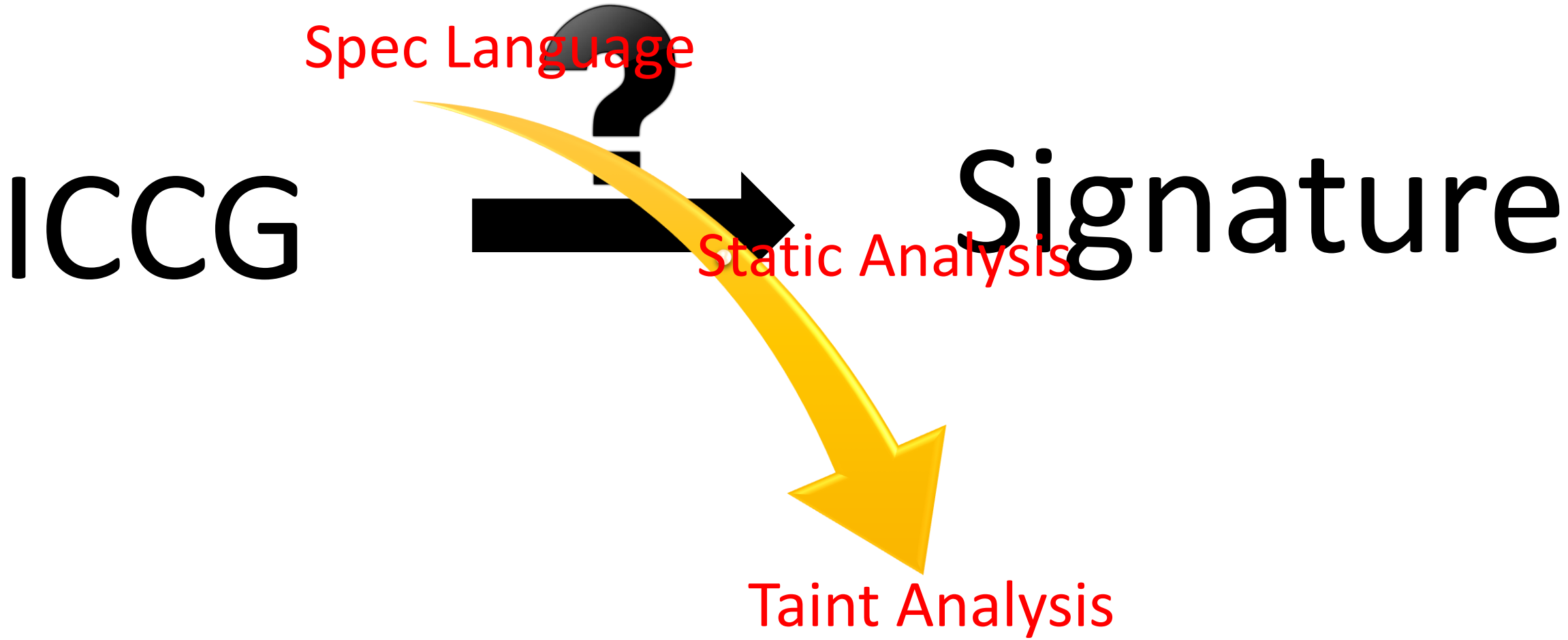
Figure 2: GoldDream signature (simplified)

Listen to the events

receiver

System invokes r when e occurs

Data flow query
Source(ID) ->
Sink(Internet)


Spec Language
ICCG
Signature
Static Analysis
Taint Analysis

Malware Spec Language

Purpose: use a languages/semantics to describe the app's inner-property/behavior

- Component type predicates: service(c)
- Predicate icc: icc*(p,q) <example>
- Predicate calls: calls(c,m)
- Predicate flows: flow(p, so, q, si)

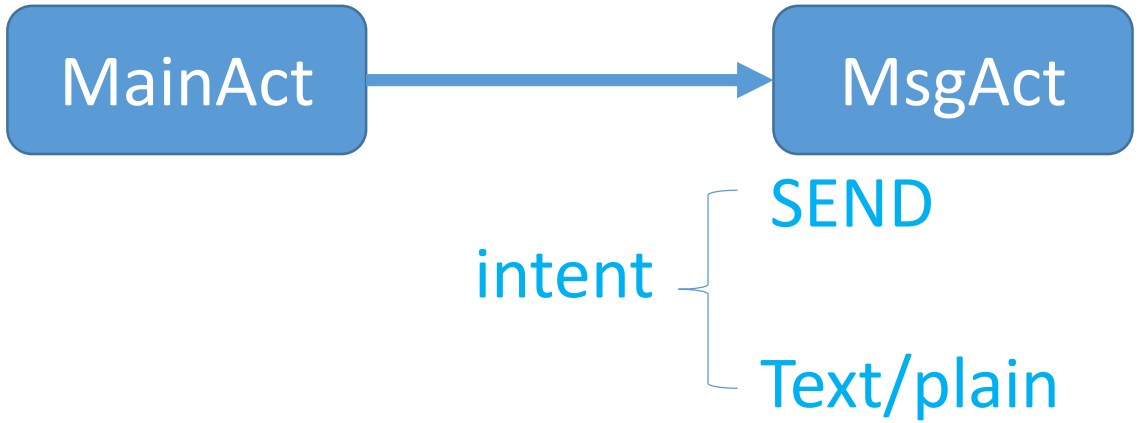
```
1. public class MainAct extends Activity {
2.     protected void onCreate(Bundle b) {
3.         foo();
4.         bar();
5.     }
6.     void foo() {
7.         Intent i = new Intent();
8.         i.setAction(android.intent.action.SEND);
9.         i.setType("text/plain");
10.        startActivity(i);
11.    }
12.     void bar() {
13.        Intent n = new Intent();
14.        n.setClass(MsgAct.class);
15.        startActivity(n);
16.    }
17. }
18. public class MsgAct extends Activity { ... }
```

Figure 3: ICC example.

```
<activity android:name="MsgAct">
  <
</activity>
```

Figure 4: A snippet of AndroidManifest.xml

Initialize an ICC through an intent



Q: Explicit intent or implicit intent?

Icc(MainAct,MsgAct,SEND,text/plain)

Static Analysis

- Call graph construction
  - conventional approach
  - pointer analysis (heap object)
- Data flow Analysis for intents <example>
  - intent analysis (intent filters)
  - transfer functions (complex algebra)
- Construct the ICCG
  - define construction rules (algebra)

```
1. public class MainAct extends Activity {
2.     protected void onCreate(Bundle b) {
3.         foo();
4.         bar();
5.     }
6.     void foo() {
7.         Intent i = new Intent();
8.         i.setAction(android.intent.action.SEND);
9.         i.setType("text/plain");
10.        startActivity(i);
11.    }
12.    void bar() {
13.        Intent n = new Intent();
14.        n.setClass(MsgAct.class);
15.        startActivity(n);
16.    }
17. }
18. public class MsgAct extends Activity { ... }
```

Figure 3: ICC example.

implicit

$$\Gamma(i_t) = \{\bot\}$$
$$\Gamma(i_a) = \{\text{action.SEND}\}$$
$$\Gamma(i_d) = \{\texttt{text/plain}\}$$

explicit

$$\Gamma(n_t) = \{\texttt{MsgAct}\}$$
$$\Gamma(n_a) = \{\bot\}$$
$$\Gamma(n_d) = \{\bot\}$$

Intent analysis

Taint Analysis

Source anno

Sink anno

```
1.  //Source annotation in android.telephony.TelephonyManager
2.  @Flow(from="$getDeviceId",to="@return")
3.  String getDeviceId(){ ... }

7.  //Sink annotation in android.telephony.SmsManager
8.  @Flow(from="text",to="!sendTextMessage")
9.  void sendTextMessage(...,String text,...){ ... }

10. //Transfer annotation in java.lang.String
11. @Flow(from="this",to="@return")
12. @Flow(from="s",to="@return")
13. String concat(String s){ ... }
```

Figure 9: Source, Sink and Transfer annotations.

If para S is tainted, Then, @return is tainted

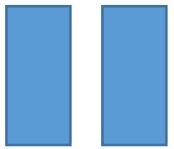
Add annotations

Taint Analysis

$$\frac{src(m_i, l),\ m_i \hookrightarrow o}{tainted(o, l)} \quad \text{(Source)}$$

$$\frac{tainted(o_1, l),\ m_i \hookrightarrow o_1,\ m_j \hookrightarrow o_2 \quad transfer(m_i, m_j)}{tainted(o_2, l)} \quad \text{(Transfer)}$$

$$\frac{tainted(o, so), m_i \hookrightarrow o, sink(m_i, si)}{flow(so, si)} \quad \text{(Sink)}$$



Static taint analysis (complex algebra)

Taint Analysis

```
public class ListDevice extends Activity {
    protected void onCreate(Bundle bd) {
1.    Device n,m;
2.      ...
3.    String x = "deviceId=";(O1)
4.    String y = TelephonyManager.g source eId(); (O2)
5.    String z = x.concat(y);(O3)
6.    m.f = z;
7.    n = m;
8.    String v = n.f;
9.    smsManager.sendTextMessage("3452",null,v,null,null);
    }
}
```

Figure 5: Example illustrating data flow

$get...Id (y) ->
O2 -> O3
-> m -> n -> v

Tainted(v, $get...Id)
Sink(send..., !send...)

flow($getDeviceId, !sendTextMessage)

Results

Table 6: Examples of Apposcopy's signatures.

Malware family	Signature
ADRD	ADRD :- receiver(r), icc(SYSTEM, r, BOOT_COMPLETED, _),receiver(s), service(t),icc*(r,s), icc*(s,t), icc*(t,s), flow(t, DeviceId, t, ENC), flow(t, SubscriberId, t, ENC), flow(t, ENC, t, Internet).
BeanBot	BeanBot :- receiver(r), service(s), service(t), service(q), icc(SYSTEM, r, PHONE_STATE, _), calls(r, abortBroadcast), icc*(r, s), icc*(s, t), icc*(s, q), flow(s, DeviceId, s, Internet), flow(s, Line1Number, s, Internet), flow(s, SimSerialNumber, s, Internet).
CoinPirate	CoinPirate :- receiver(r), receiver(t), icc(SYSTEM, r, SMS_SENT, _), icc(SYSTEM, r, SMS_RECEIVED, _), service(s), calls(r, abortBroadcast), calls(s, sendTextMessage), icc*(r, s), icc*(s, t), flow(s, DeviceId, s, Internet), flow(s, SubscriberId, s, Internet), flow(s, Model, s, Internet), flow(s, SDK, s, Internet).

Describe components/flows/ICCs

Results

FN = A belongs to family F but Apposcopy cannot detect

FP = A does not belong to family F but Apposcopy wrongly identifies

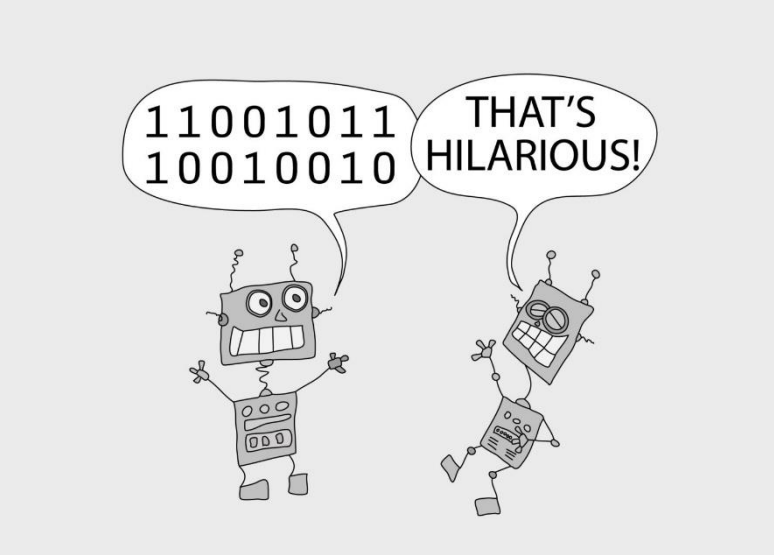
Why FN >> FP?

Table 7: Evaluation of Apposcopy on malware from the Android Malware Genome project.

Malware Family	#Samples	FN	FP	Accuracy
DroidKungFu	444	15	0	96.6%
AnserverBot	184	2	0	98.9%
BaseBridge	121	75	0	38.0%
Geinimi	68	2	2	97.1%
DroidDreamLight	46	0	0	100.0%
GoldDream	46	1	0	97.8%
Pjapps	43	7	0	83.7%
ADRD	22	0	0	100.0%
jSMSHider	16	0	0	100.0%
DroidDream	14	1	0	92.9%
Bgserv	9	0	0	100.0%
BeanBot	8	0	0	100.0%
GingerMaster	4	0	0	100.0%
CoinPirate	1	0	0	100.0%
DroidCoupon	1	0	0	100.0%
Total	1027	103	2	90.0%

Q&A

Why semantic-based?



behavior

behavior signature

01010010010

Bytecode signature

Can apposcopy identify malicious/benign behavior? No

Signature resistance?

Code reordering/code injection/code rewriting