CS 5430

Information Flow in Android Apps

Prof. Clarkson
Spring 2016

ClickRelease

- Prototype tool [Micinski, Fetter-Degges, Jeon, Foster, Clarkson 2015]
- Checks whether Android apps obey users' intent when declassifying confidential information
  - **Intent** expressed through GUI interactions
  - **Declassification** policies: based on formal logic
  - **Information** could include contact details, GPS location, ...
- Focus is on the **user** not the program

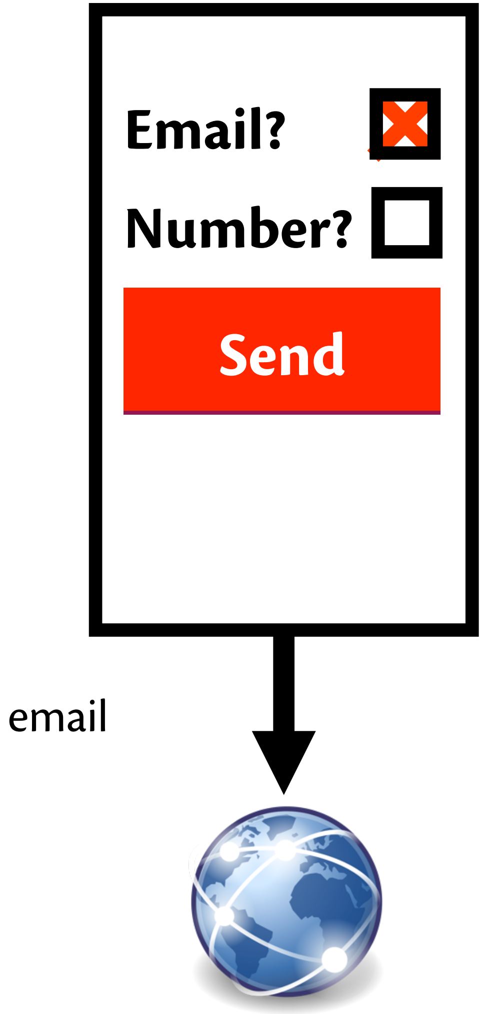
Android

- Popular mobile platform
- Authorization regulated with permissions
  - e.g., camera, read contacts, write contacts, access fine location, access coarse location, read phone state, write call log, ...
  - Specified by developer
  - Requested from user during installation (before Android 6.0 Oct 2015)

Permissions

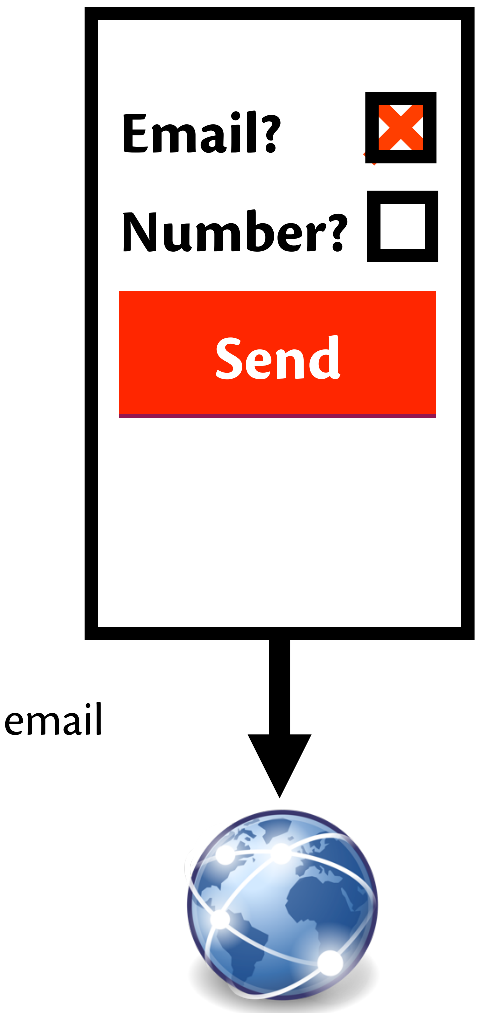
- Weaknesses:
  - **Trojan horse:** app maliciously requests permissions it doesn't need, user grants, app abuses permission
  - **Programmer mistakes:** app wrongly releases user's sensitive information
- Permissions provide access control not information-flow control
- Control access to a resource, not usage of information from that resource

Bump app



- User checks “Email”
- Clicks "Send"
- App sends user's email address over network

Bump app – Buggy or malicious



- User checks “Email”
- Clicks "Send"
- App sends user's phone number over network
- Worse yet: app sends all the user's private contact information over network
- Not the user's intent

Our solution

- **Policies** for capturing user intent
- **Formal security condition** called Interaction-Based Noninterference (IBNI)
- **Prototype tool** ClickRelease that checks Android apps
- **Evaluation** of some apps and policies

POLICIES

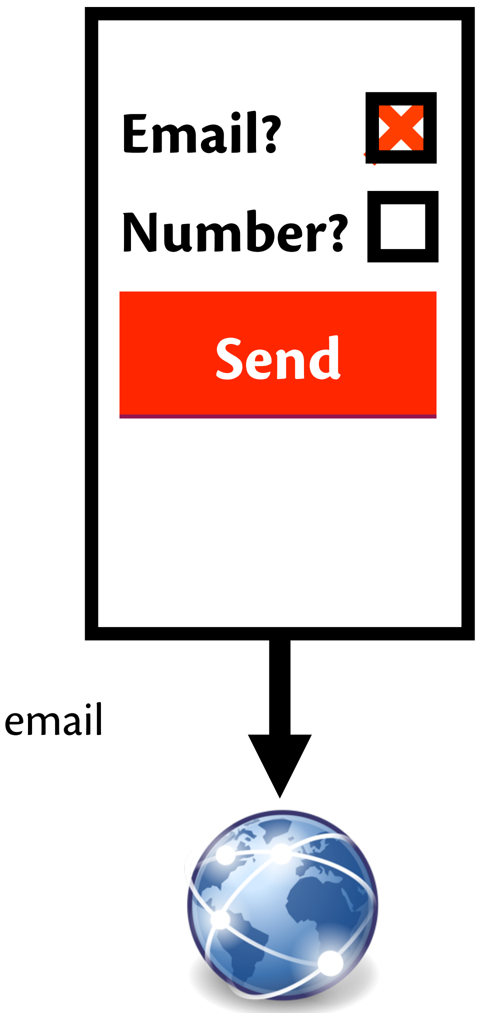
Declassification policies

- **Core idea:**
  - GUI interactions generate events
  - Events have security level: public, secret, ...
  - Use a temporal logic to specify when an event may be declassified to lower level because of user intent
- **Security analyst** writes these policies extentionally:
  - Not embedded in source code
  - But tied to the events generated by the code

Events

- **Security-relevant actions** taken by user and app
  - GUI interactions: buttons, check boxes, ...
  - Writes and reads by app: network, stored data, ...
- Each event comprises channel and value
- In source code, correspond to method calls
  - GUI: handler registered to receive callback
  - Write and reads: API calls
- Execution of app produces many such method calls
- We abstract them to an event trace...

Event trace for bump app



- App initializes, reads contacts
- User checks “Email”
- Clicks "Send"
- App sends user's email address over network

```
email : clarkson@cs.cornell.edu
phone : 607-255-0278
emailBox : true
sendButton : unit
netout : clarkson@cs.cornell.edu
```

Event security

- Security level: how confidential event is (could be a lattice)
- **Threat model**:
  - Public events may be revealed to attacker
  - Secret events may not
  - Attacker's only means to observe app is network, so writes to `netout` are public
- Policy determines security level of event... (default: secret)

Policies

Examples:

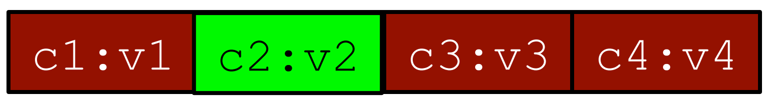
1. **Bump app:** Phone number may be revealed when Send button is clicked if phone number checkbox is checked
2. **Contact picker:** Currently selected contact from a spinner may be revealed, but no others
3. **Location toggle:** Phone's fine-grained GPS location may be revealed when fine-grained checkbox is checked; otherwise, coarse-grained location may be revealed
4. **WhereRU:** Phone's location may be revealed always, never, or on demand, based on chosen radio button

Common element: ordering of events...

Policies

Policy: form @ lvl

- Formula form identifies an event in a trace
- Policy stipulates security level lvl of **that** event
- e.g., "second event" @ public



Formulas:

- based on quantified linear-time temporal logic (QTL) [Lichtenstein et al. 1985]
- customized to GUI interactions

Our temporal logic

$$
\begin{array}{lll}
\phi & ::= & e \\
 & & \mid \neg\phi \mid \phi_1 \wedge \phi_2 \mid \phi_1 \vee \phi_2 \\
 & & \mid X\,\phi \mid F\,\phi \mid G\,\phi \mid P\,\phi \mid \phi_1\, U\, \phi_2 \mid \phi_1\, S\, \phi_2 \\
 & & \mid \forall x \,.\, \phi \mid \exists x \,.\, \phi \\
e & ::= & \boldsymbol{\mathtt{\mathit{name}}}\ \mathtt{:}\ t \\
t & ::= & x \mid v \\
v & ::= & \mathit{int} \mid \mathbf{\mathtt{true}} \mid \mathbf{\mathtt{false}} \mid \mathbf{\mathtt{unit}}
\end{array}
$$

Our temporal logic

Boolean connectives

$$
\begin{array}{lll}
\phi & ::= & e \\
 & & \mid \neg\phi \mid \phi_1 \wedge \phi_2 \mid \phi_1 \vee \phi_2 \\
 & & \mid \mathrm{X}\,\phi \mid \mathrm{F}\,\phi \mid \mathrm{G}\,\phi \mid \mathrm{P}\,\phi \mid \phi_1\,\mathrm{U}\,\phi_2 \mid \phi_1\,\mathrm{S}\,\phi_2 \\
 & & \mid \forall x\,.\,\phi \mid \exists x\,.\,\phi \\
e & ::= & \mathtt{\textit{name}} \;\mathtt{:}\; t \\
t & ::= & x \mid v \\
v & ::= & \mathit{int} \mid \mathtt{\textbf{true}} \mid \mathtt{\textbf{false}} \mid \mathtt{\textbf{unit}}
\end{array}
$$

Our temporal logic

Temporal connectives

$$
\begin{aligned}
\phi \quad & ::= \; e \\
& \quad \mid \neg\phi \mid \phi_1 \wedge \phi_2 \mid \phi_1 \vee \phi_2 \\
& \quad \mid X\,\phi \mid F\,\phi \mid G\,\phi \mid P\,\phi \mid \phi_1\, U\, \phi_2 \mid \phi_1\, S\, \phi_2 \\
& \quad \mid \forall x\,.\,\phi \mid \exists x\,.\,\phi \\
e \quad & ::= \mathtt{\textit{name}} \; \mathtt{:} \; t \\
t \quad & ::= x \mid v \\
v \quad & ::= \mathit{int} \mid \mathtt{true} \mid \mathtt{false} \mid \mathtt{unit}
\end{aligned}
$$

Temporal connectives

Connective	Meaning
X ϕ	ϕ will be true **next**
F ϕ	ϕ will hold in the **future** (at some time)
P ϕ	ϕ held in the **past** (at some time)
G ϕ	ϕ holds **globally** (at all times in the future)
ϕ U ψ	ϕ will be true **until** ψ is true
ϕ S ψ	ϕ has been true **since** ψ was true

Our temporal logic

$$
\begin{array}{lll}
\phi & ::= & e \\
 & & \mid \neg\phi \mid \phi_1 \wedge \phi_2 \mid \phi_1 \vee \phi_2 \\
 & & \mid \mathrm{X}\,\phi \mid \mathrm{F}\,\phi \mid \mathrm{G}\,\phi \mid \mathrm{P}\,\phi \mid \phi_1\,\mathrm{U}\,\phi_2 \mid \phi_1\,\mathrm{S}\,\phi_2 \\
 & & \mid \forall x\,.\,\phi \mid \exists x\,.\,\phi \\
e & ::= & \mathtt{\textit{name}} \;\mathtt{:}\; t \\
t & ::= & x \mid v \\
v & ::= & \mathit{int} \mid \mathtt{true} \mid \mathtt{false} \mid \mathtt{unit}
\end{array}
$$

Quantifiers over terms

Our temporal logic

$$
\begin{array}{lll}
\phi & ::= & e \\
 & & \mid \neg\phi \mid \phi_1 \wedge \phi_2 \mid \phi_1 \vee \phi_2 \\
 & & \mid X\,\phi \mid F\,\phi \mid G\,\phi \mid P\,\phi \mid \phi_1\,U\,\phi_2 \mid \phi_1\,S\,\phi_2 \\
 & & \mid \forall x\,.\,\phi \mid \exists x\,.\,\phi \\
e & ::= & \mathbf{\mathit{name}} \; : \; t \\
t & ::= & x \mid v \\
v & ::= & \mathit{int} \mid \mathtt{true} \mid \mathtt{false} \mid \mathtt{unit}
\end{array}
$$

Quantifiers over terms

which are program values

Our temporal logic

event

$$
\begin{array}{lll}
\phi & ::= & e \\
 & & \mid \neg\phi \mid \phi_1 \wedge \phi_2 \mid \phi_1 \vee \phi_2 \\
 & & \mid X\,\phi \mid F\,\phi \mid G\,\phi \mid P\,\phi \mid \phi_1\, U\, \phi_2 \mid \phi_1\, S\, \phi_2 \\
 & & \mid \forall x\,.\,\phi \mid \exists x\,.\,\phi \\
e & ::= & \texttt{name} : t \\
t & ::= & x \mid v \\
v & ::= & \mathit{int} \mid \texttt{true} \mid \texttt{false} \mid \texttt{unit}
\end{array}
$$

Our temporal logic

Other extensions:

- Wildcard term **`*`**

  `chan:*` is any event on **`chan`**

- Last event on a channel

  last(**`chan`**, t) means last event on **`chan`** had value t

- *rest not important here*

Bump app policy

Intuition: phone number may be revealed when Send button is clicked if phone number checkbox is checked

`phone:*`

∧ F(**`sendButton:unit`**

∧ last(**`phoneBox`**,**`true`**))

@ public

Bump app policy

Intuition: phone number may be revealed when Send button is clicked if phone number checkbox is checked

If current input is phone number...

phone:*

∧ F(**sendButton:unit**

∧ last(**phoneBox**,**true**))

@ public

Bump app policy

Intuition: phone number may be revealed when Send button is clicked if phone number checkbox is checked

If current input is phone number...

...and eventually send button clicked...

`phone:*`

∧ F(`sendButton:unit`

∧ last(`phoneBox,true`))

@ public

Bump app policy

Intuition: phone number may be revealed when Send button is clicked if phone number checkbox is checked

If current input is phone number...

...and eventually send button clicked...

...and at that point phone number box checked...

```
phone:*
∧ F(sendButton:unit
      ∧ last(phoneBox,true))
@ public
```

Bump app policy

Intuition: phone number may be revealed when Send button is clicked if phone number checkbox is checked

If current input is phone number...

$\texttt{phone:*}$
$\wedge\ F(\texttt{sendButton:unit}$
$\quad\wedge\ \text{last}(\texttt{phoneBox,true}))$
@ public

...and eventually send button clicked...

...and at that point phone number box checked...

...then value of phone number is public.

Bump app policy

Intuition: phone number may be revealed when Send button is clicked if phone number checkbox is checked

`phone:*`

∧ F(**`sendButton:unit`**

∧ last(**`phoneBox`**,**`true`**))

@ public

but **other events remain secret** (e.g., email)

Bump app policy

`phone:*`
∧ F(**`sendButton:unit`**
∧ last(**`phoneBox`**,**`true`**))
@ public

Constrains **when** secret information is read:

- If phone number read **after** button clicked,
- then formula would **not** hold,
- hence security level remains **secret**

Location app policy

- **Intuition:** Phone's fine-grained GPS location may be revealed when fine is checked; otherwise, coarse-grained location may be revealed
- Coarse-grained: **mask** lower 8 bits

Location app policy

`gps:*` ∧ last(**`radio`**,**`"fine"`**) @ public,

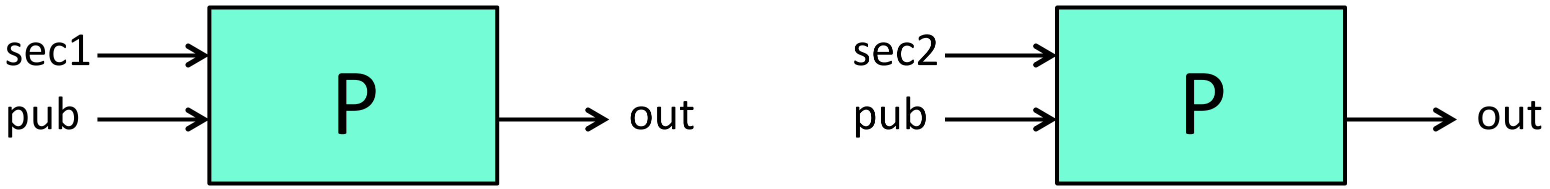
`gps:*` ∧ last(**`radio`**,**`"coarse"`**) @ mask

- **set** of policies
- security level mask between public and secret
  - characterizes what attacker may observe
  - *security condition* makes use of level...

SECURITY CONDITION

Security condition

- **Noninterference** [Goguen and Meseguer 1982]: *actions of high-security users do not affect observations of low-security users*
- Intuition, as commonly adapted to programs: *changes to secret inputs do not cause observable change in public output*



Security condition

Interaction-based noninterference (IBNI)

- Our new noninterference property
- **Intuition:** *two event traces with the same secret input events have the same public output events*
- Builds on observational determinism [Zdancewic and Myers 2003]

Security condition

Interaction-based noninterference (IBNI)

Toward a formal definition:

- Represent program as a set T of event traces; formal semantics defines that set
- Define function label(t,pol) to label each event in trace with its security level according to policy pol
- Define equivalence relation $\equiv_S$ on labeled traces: $t1 \equiv_S t2$ if observer cleared at level S perceives traces as having the same events
- Define function inputs(t) to project out only the input events from a trace (labeled or unlabeled)

IBNI

Definition of IBNI:

Program T satisfies IBNI for security policy pol if:

for all traces t1 and t2 in T, and for all security levels S,

letting l1=label(t1,pol) and l2=label(t2,pol),

it holds that

inputs(l1) $\equiv_S$ inputs(l2) implies l1 $\equiv_S$ l2.

Structure of this definition is entirely standard

Interesting part is label...

IBNI

label(t, pol) =

(t[0], level(t, pol, 0)), (t[1], level(t, pol, 1)), ...

level(t, pol, i) =

if t[i] = netout : p then public

else

form the set of all levels S

such that f@S in pol and f holds at t[i];

return the lowest-security element of that set

IBNI

label(t, pol) =

$$(t[0], \text{level}(t, pol, 0)), (t[1], \text{level}(t, pol, 1)), \ldots$$

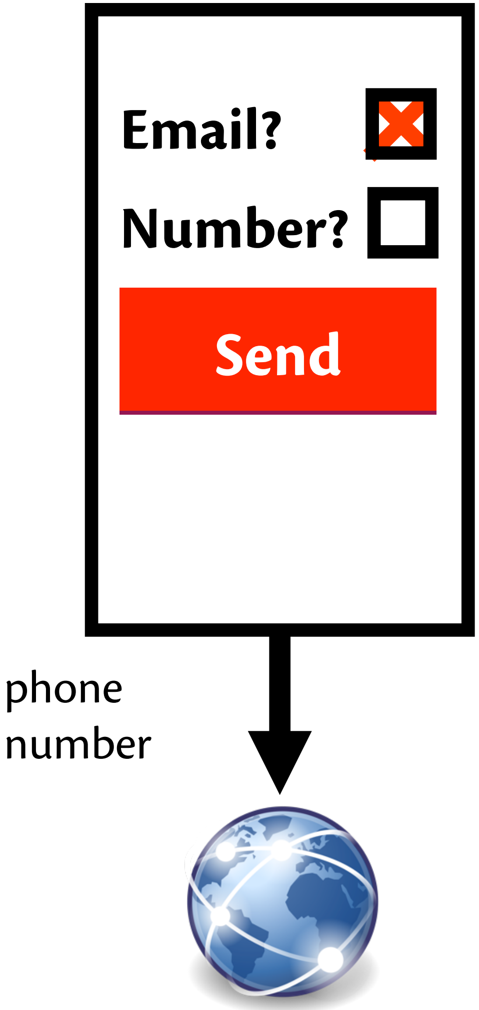
level(t, pol, i) =

$$\text{if } t[i] = \text{netout} : p \text{ then public}$$

$$\text{else } \sqcap_{\phi@S \in pol} \{S \mid t,i \vDash \phi\}$$

relation $\vDash$ is essentially standard QTL

IBNI with insecure bump app



Insecure variant of bump app:

- releases phone number when email address checked
- and vice-versa

IBNI with insecure bump app

Two possible traces:

- **email:a@b.com, phone:202-555-0000, phoneBox:false, emailBox:true, sendButton:unit, netout:202-555-0000**
- **email:a@b.com, phone:202-555-1337, phoneBox:false, emailBox:true, sendButton:unit, netout:202-555-1337**

Ok to reveal these GUI events

Policy:

phone:* ∧ F(**sendButton:unit** ∧ last(**phoneBox**,**true**)) @ public

email:* ∧ F(**sendButton:unit** ∧ last(**emailBox**,**true**)) @ public

phoneBox:* @ public, **emailBox:*** @ public, **sendButton:*** @ public

IBNI with insecure bump app

Two possible traces:

- `email:a@b.com, phone:202-555-0000, phoneBox:false, emailBox:true, sendButton:unit, netout:202-555-0000`
- `email:a@b.com, phone:202-555-1337, phoneBox:false, emailBox:true, sendButton:unit, netout:202-555-1337`

Policy:

phone:* ∧ F(**sendButton:unit** ∧ last(**phoneBox**,**true**)) @ public,

email:* ∧ F(**sendButton:unit** ∧ last(**emailBox**,**true**)) @ public,

phoneBox:* @ public, **emailBox:*** @ public, **sendButton:*** @ public

Labeling: **netout** and GUI events are public, but **phone** and **email** aren't

IBNI with insecure bump app

Two possible traces:

- `email:a@b.com, phone:202-555-0000,`
  `phoneBox:false, emailBox:true,`
  `sendButton:unit, netout:202-555-0000`
- `email:a@b.com, phone:202-555-1337,`
  `phoneBox:false, emailBox:true,`
  `sendButton:unit, netout:202-555-1337`

IBNI: not satisfied

- two traces
- same public inputs
- different public outputs

IBNI

Definition of IBNI:

Program T satisfies IBNI for security policy pol if:

for all traces t1 and t2 in T, and for all security levels S,

letting l1=label(t1,pol) and l2=label(t2,pol),

it holds that

inputs(l1) $\equiv_S$ inputs(l2) implies l1 $\equiv_S$ l2.

PROTOTYPE

Prototype tool

ClickRelease:

- Our implementation of IBNI checking for Android
- Based on SymDroid [Jeon et al. 2012]: symbolic executor for Dalvik bytecode
- Itself based on Z3 [de Moura and Bjørner 2008]: SMT solver

Symbolic execution

[Clarke 1976, King 1976]

- Motivated by software testing:
  - Goal is to check programs for presence of errors
  - And generate inputs that would trigger errors
  - Errors can be debugged and fixed
- Key idea: symbolic values
  - e.g. α instead of 5
  - Program variables and expressions can be symbolic
- Symbolic executor explores all paths of program execution
  - Execution path: the sequence of branches taken during execution
  - Goal is to find a concrete input that triggers each possible execution path
  - Might not be complete: explore up to some resource bound

Symbolic execution

- Maintain a list of program states each of which corresponds to a particular point of execution
- State comprises:
  - memory: maps variables, heap locations, etc. to symbolic values
  - path condition: logical formula that captures what branches have been taken to reach current program point
  - program counter: next statement to execute
- Start with a single state (initial memory, path condition is simply true)

Symbolic execution algorithm

- Take a state off the list
- Execute the next program statement
  - **Assignment:** update memory with symbolic result, add resulting state back to the list
  - **If statement** with guard e: add two states back to the list
    - one has path condition updated with "and e"
    - other updated with "and not e"
  - **Loops:** can lead to infinite number of paths to explore; must bound somehow (timeout, iterations, exploration depth, etc.)
  - **Function calls:** need code, specification, or must treat symbolically

Symbolic execution algorithm

- If path condition ever becomes unsatisfiable, no reason to explore further; terminate along that path
- If program exits or encounters error:
  - Symbolic execution terminates
  - Path condition sent to satisfiability solver to find concrete inputs that would lead to that path

Symbolic execution of Android

- SymDroid [Jeon et al. 2012]:
  - Java source code of Android apps compiled into Dalvik bytecode
  - SymDroid is symbolic executor for Dalvik
- Android is more than just bytecode:
  - **Libraries**, some written in native code
  - **System services** (telephony, GPS, etc.)
  - **Entry points** and callbacks into apps (apps register components that respond to Intents – not just a main function)

Symbolic execution of Android

- SymDroid **models** instead of executing Android platform code
  - Model can be written in Java or in OCaml (SymDroid's source language)
  - Handles about 25% of the Android Compatibility Test Suite (CTS); failed cases are all because of unmodeled system libraries; open challenge how to fully model
- Model includes:
  - Generating clicks in GUI
  - GUI events from widgets (buttons, check boxes, etc.)
  - Services (telephony, GPS, etc.)

Symbolic execution of GUI

- **Problem:**
  - Not just a single input at beginning of execution
  - Instead, apps receive streams of inputs from user
  - So need to simulate user
- **Solution:**
  - Custom driver for each app
  - Calls methods in Android model to inject GUI events
  - Driver runs a loop that nondeterministically picks a new event to inject
  - Performance of symbolic execution is exponential in input depth: number of iterations of loop

EVALUATION

Apps

1. **Bump app:** Phone number may be revealed when Send button is clicked if phone number checkbox is checked

   Insecure variants: release email instead; always release phone after three more clicks

2. **Contact picker:** Currently selected contact from a spinner may be revealed, but no others

   Insecure variants: scan contact list to release particular one in addition to selected contact; release different contact than selected

Apps

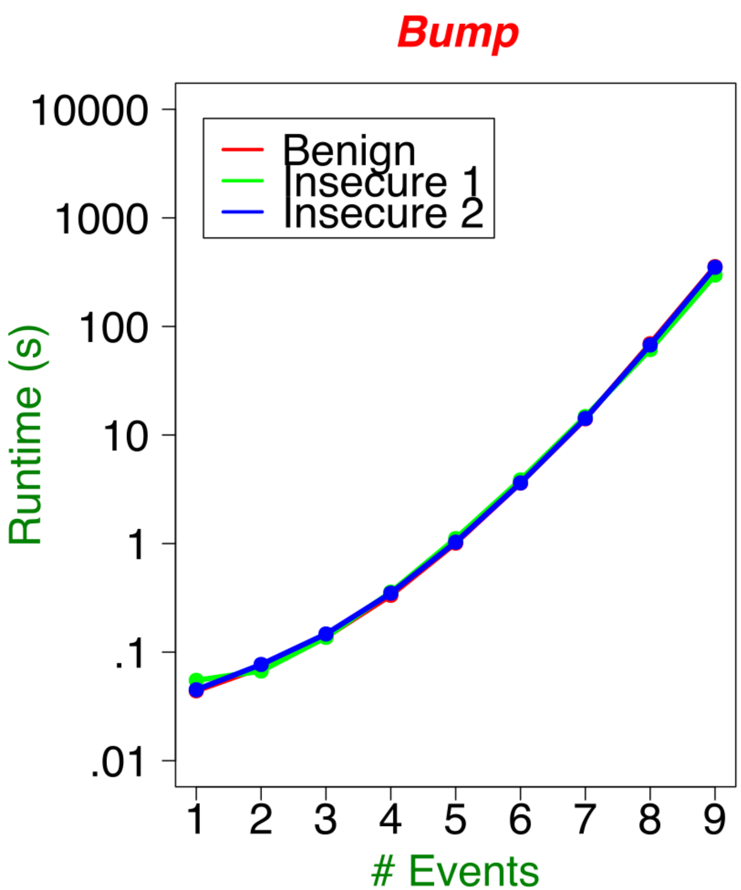
3. **Location toggle:** Phone's fine-grained GPS location may be revealed when fine-grained checkbox is checked; otherwise, coarse-grained location may be revealed

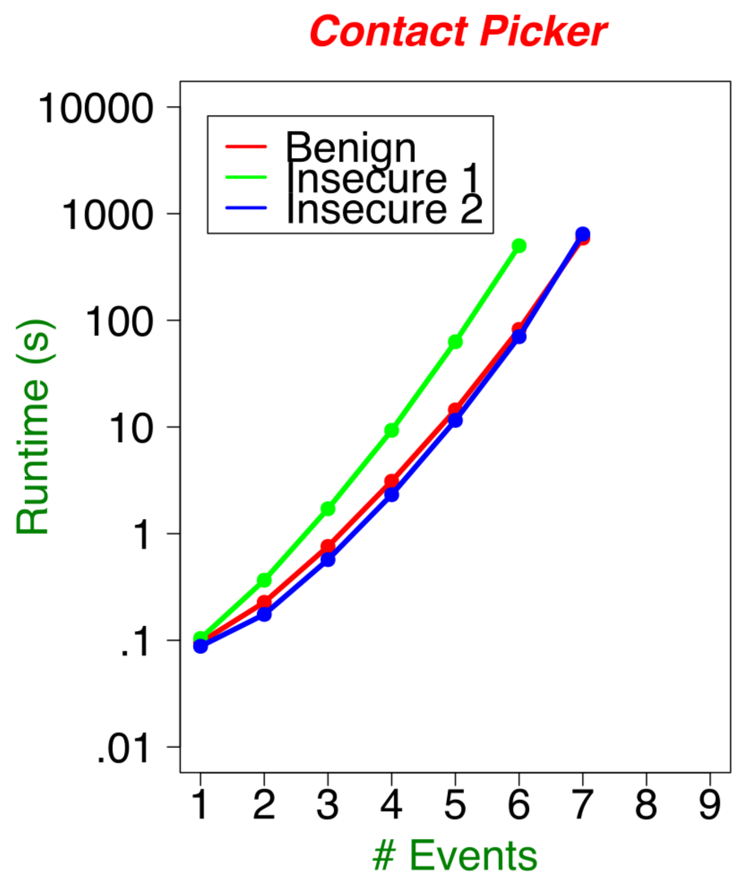
   Insecure variants: always release fine-grained; store fine-grained and release it later even if coarse checked

4. **WhereRU:** Phone's location may be revealed always, never, or on demand, based on chosen radio button

   Insecure variants: always share regardless; share location from past when choice might have been different

Scalability





(4-core i7 CPU @3.5GHz, 16GB RAM, Ubuntu 14, median of 10 runs)

For four apps, can explore input depth of 5-9 events within an hour

Scalability

- **Small counter model hypothesis:** if there are bugs, they are likely to be revealed by some short sequence of inputs
- Holds for our apps: need only 2-5 inputs for each to reveal an illegal information flow
- And we can completely explore that space within an hour
- So even though complexity is exponential, finding security violations is relatively efficient
- Scaling up? Larger apps will need:
  - more complete Android model
  - larger counterexamples

CONCLUSION

Summary

- **Policies** for capturing user intent
- **Formal security condition** called Interaction-Based Noninterference (IBNI)
- **Prototype tool** ClickRelease that checks Android apps
- **Evaluation** of some apps and policies

Related work

- Access control gadgets [Roesner et al. 2012]
- AppIntent [Yang et al. 2013]
- Pegasus [Chen et al. 2013]
- DIFC for Android [Jia et al. 2013]
- SIF [Chong et al. 2007]
- Cassandra [Lortz et al. 2014]
- Declassification policies [Chong and Myers 2004]

Upcoming events

- [Sunday] A6 due
- [May 16] Final exam

If secrecy is the beginning of tyranny, declassification is its apotheosis. – John Alejandro King