DANIEL BARTON

FLOWDROID: PRECISE CONTEXT, FLOW FIELD, OBJECT-SENSITIVE AND LIFECYCLE AWARE TAINT ANALYSIS FOR ANDROID APPS

PAPER BACKGROUND

- Authors: Steven Arzt, Siegfried Rastloser, Christian Fritz, Eric Bodden, et al. (Damien Octeau)
- ACM SIGPLAN conference on Programming Language Design and Implementation (18%).

FLOWDROID OVERVIEW

- Novel static taint-analysis system tailored for Android.
- Analyzes both app byte-code and configuration files.
- First context-, flow-, field-, object-sensitive taint analysis.
    - On-demand alias analysis to support context and object sensitivities, based on Andromeda.
- Use cases: secure Android apps, identify Android Malware

ATTACK/THREAT MODEL

- FlowDroid will detect tainted flows regardless of malice.
- Attacker supplies arbitrary byte-code.
- Goal: Leak private data.
- Attacker cannot circumvent Android security or use side channels.
- Conforms to standard malware.

ANDROID OVERVIEW

- Android app != Java program
  - Multiple points of entry
- Components
  - Activities - Screens
  - Services - Background operations
  - Content Providers - Database-like storage
  - Broadcast Receivers - Global event listeners

```
1  public class LeakageApp extends Activity{
2  private User user = null;
3  protected void onRestart(){
4    EditText usernameText =
       (EditText)findViewById(R.id.username);
5    EditText passwordText =
       (EditText)findViewById(R.id.pwdString);
6    String uname = usernameText.toString();
7    String pwd = passwordText.toString();
8    if(!uname.isEmpty() && !pwd.isEmpty())
9      this.user = new User(uname, pwd);
10 }
11 //Callback method in xml file
12 public void sendMessage(View view){
13   if(user == null) return;
14   Password pwd = user.getpwd();
15   String pwdString = pwd.getPassword();
16   String obfPwd = "";
17   //must track primitives:
18   for(char c : pwdString.toCharArray())
19     obfPwd += c + "_"; //String concat.
20
21   String message = "User: " +
22      user.getName() + " | Pwd: " + obfPwd;
23   SmsManager sms = SmsManager.getDefault();
24   sms.sendTextMessage("+44 020 7321 0905",
25     null, message, null, null);
26 }
```

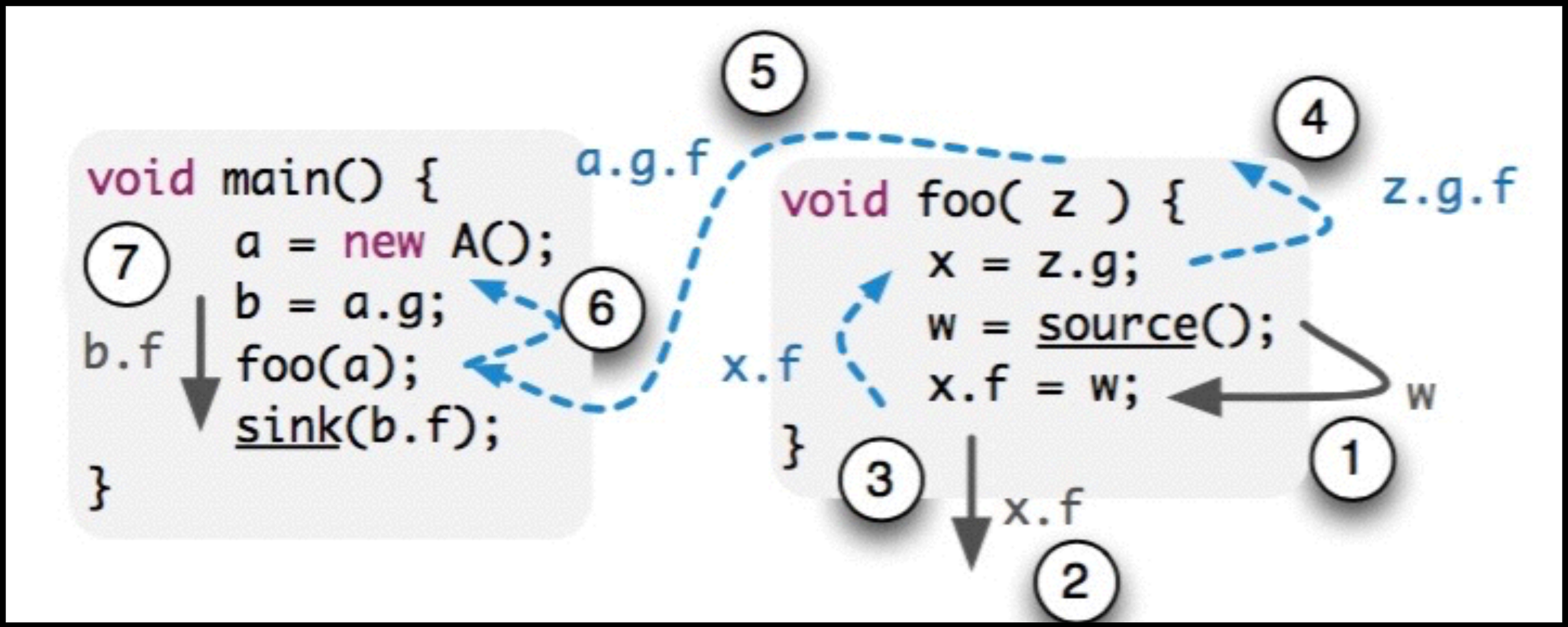
ANDROID CONTROL/ DATA FLOW GRAPH

FLOWDROID MODEL OF APP LIFECYCLE

- Assumes components can execute in an arbitrary sequential order.
- Based on IFDS analysis, path insensitive.
- Solution: Generate *dummy* main method.
    - Each path is possible, does not traverse all paths.
    - Callbacks only analyzed during execution windows in parent component. Scans XML files, generates call graph per lifecycle method.
- Generates final call graph with dummy method as entry point.

FLOWDROID TAINT ANALYSIS

- Combines forward taint analysis and on-demand backward aliasing.



TAINT ANALYSIS

- Access paths
  - x.f.g
  - Configurable lengths (5 by default)
    - Includes all possible paths (x.f = x.f.g, x.f.h)
- Transfer Function
  - Taints left side if the operands on right are tainted.

ON-DEMAND ALIAS ANALYSIS

- When a tainted value is assigned to the heap, search backward for aliases and taint them as well.

- Perform forward taint propagation for each found alias.

- Problem: Produces unrealizable paths along conflicting contexts when used together (i.e. context insensitive results).

  - Solution: Inject forward analysis context into backward analysis.

Algorithm 1 Main loop of forward solver

1: **while** $\mathit{WorkList}_{FW} \neq \emptyset$ **do**
2: pop $\langle s_p, d_1 \rangle \rightarrow \langle n, d_2 \rangle$ off $\mathit{WorkList}_{FW}$
3: **switch** (n)
4: **case** n is call statement:
5: **if** summary exists for call **then**
6: apply summary
7: **else**
8: map actual parameters to formal parameters
9: **end if**
10: **case** n is exit statement:
11: install summary $\langle s_p, d_1 \rangle \rightarrow \langle n, d_2 \rangle$
12: map formal parameters to actual parameters
13: map return value back to caller's context
14: **case** n is assignment $lhs = rhs$:
15: $d_3 :=$ replace rhs by lhs in d_2
16: insert $\langle s_p, d_1 \rangle \rightarrow \langle n, d_3 \rangle$ into $\mathit{WorkList}_{BW}$
17: extend path-edges via the *propagate*-method of the classical IFDS algorithm
18: **end while**

Algorithm 2 Main loop of backward solver

1: **while** $\mathit{WorkList}_{BW} \neq \emptyset$ **do**
2: pop $\langle s_p, d_1 \rangle \rightarrow \langle n, d_2 \rangle$ off $\mathit{WorkList}_{BW}$
3: **switch** (n)
4: **case** n is call statement:
5: **if** summary exists for call **then**
6: apply summary
7: **else**
8: map actual parameters to formal parameters
9: **end if**
10: extend path-edges via the *propagate*-method of the classical IFDS algorithm
11: **case** n is method's first statement:
12: install summary $\langle s_p, d_1 \rangle \rightarrow \langle n, d_2 \rangle$
13: insert $\langle s_p, d_1 \rangle \rightarrow \langle n, d_2 \rangle$ into $\mathit{WorkList}_{FW}$
14: do *not* extend path-edges via the *propagate*-method of the classical IFDS algorithm, killing current taint d_2
15: **case** n is assignment $lhs = rhs$:
16: $d_3 :=$ replace lhs by rhs in d_2
17: insert $\langle s_p, d_1 \rangle \rightarrow \langle n, d_3 \rangle$ into $\mathit{WorkList}_{FW}$
18: extend path-edges via the *propagate*-method of the classical IFDS algorithm
19: **end while**

ON-DEMAND ALIAS ANALYSIS (CONT.)

- Problem: Forward/backward combination lead to flow insensitive results.
  - Solution: Augment access path with statement that spawns the backward alias, the activation statement.
- Activation statements are used to look up call trees in which they occur.

WHY PRESERVE ALL THESE SENSITIVITIES?

- Model lifecycle accurately to reduce false negatives.
- Field sensitivity allows for reduced false positives.
- Object sensitivity to automatically dismiss false positives (i.e. when different objects hit the same code).
- Context sensitivity to eliminate unrealized paths, and reduce false positives.

FLOWDROID ARCHITECTURE

- Unzip .akp
- Search byte-code and layout XML files for life cycle methods, callbacks, sources, and sinks.
- Generate dummy main method from list of life cycle methods and call backs.
- Generate call graph and inter-procedural control flow graph (ICFG).
- Perform taint analysis on sources in ICFG.

LIMITATIONS

- Resolves reflective calls only if their arguments are string constants.

- Could miss callbacks (native methods that are not recognized as callbacks).

- Does not account for multiple threads.

EVALUATION

- Addressed 4 research questions:
  - How does FlowDroid compare to commercial taint-analysis tools for android in terms of precision and real?
  - Can FlowDroid find all privacy leaks in InsecureBank, and app specifically designed by others to challenge vulnerability detection tools for android, and what is its performance?
  - Can FlowDroid find leaks in real world apps and how fast?
  - How well does FlowDroid perform when analyzing Java programs?

EXPERIMENTAL SETUP

- DroidBench
  - 39 hand-crafted Android apps.
  - Crafted to challenge static analysis problems (different sensitivities, etc.) and Android specific challenges (modeling lifecycle).
  - First Android specific benchmark suite.

Arrays and Lists			
ArrayAccess1			★
ArrayAccess2	★	★	★
ListAccess1	★	★	★
Callbacks			
AnonymousClass1	○	⊛	⊛
Button1	○	⊛	⊛
Button2	⊛ ○ ○	⊛ ○ ○	⊛ ⊛ ⊛ ★
LocationLeak1	○ ○	○ ○	⊛ ⊛
LocationLeak2	○ ○	○ ○	⊛ ⊛
MethodOverride1	⊛	⊛	⊛
Field and Object Sensitivity			
FieldSensitivity1			
FieldSensitivity2			
FieldSensitivity3	⊛	⊛	⊛
FieldSensitivity4	★		
InheritedObjects1	⊛	⊛	⊛
ObjectSensitivity1			
ObjectSensitivity2	★		
Inter-App Communication			
IntentSink1	⊛	⊛	○
IntentSink2	⊛	⊛	⊛
ActivityCommunication1	⊛	⊛	⊛
Lifecycle			
BroadcastReceiverLifecycle1	⊛	⊛	⊛
ActivityLifecycle1	⊛	⊛	⊛
ActivityLifecycle2	○	⊛	⊛
ActivityLifecycle3	○	○	⊛
ActivityLifecycle4	○	⊛	⊛
ServiceLifecycle1	○	○	⊛
General Java			
Loop1	⊛	○	⊛
Loop2	⊛	○	⊛
SourceCodeSpecific1	⊛	⊛	⊛
StaticInitialization1	○	⊛	○
UnreachableCode		★	
Miscellaneous Android-Specific			
PrivateDataLeak1	○	○	⊛
PrivateDataLeak2	⊛	⊛	⊛
DirectLeak1	⊛	⊛	⊛
InactiveActivity	★	★	
LogNoLeak			
Sum, Precision and Recall			
⊛, higher is better	14	17	26
★, lower is better	5	4	4
○, lower is better	14	11	2
Precision $p = ⊛/(⊛ + ★)$	74%	81%	86%
Recall $r = ⊛/(⊛ + ○)$	50%	61%	93%
F-measure $2pr/(p + r)$	0.60	0.70	0.89

EVALUATION RESULTS

- Q2: 31 seconds to complete with a stock laptop, finds all vulnerabilities without false positives or false negatives.
- Q3: ran FlowDroid on 500 Google Play apps. Nothing malicious. Ran again on 1000 known malware. Averaged 2 data leaks.
- Q4: ran FlowDroid on Stanford SecuriBench (J2EE benchmark).

Test-case group	TP	FP
Aliasing	11/11	0
Arrays	9/9	6
Basic	58/60	0
Collections	14/14	3
Datastructure	5/5	0
Factory	3/3	0
Inter	14/16	0
Pred	n/a	n/a
Reflection	n/a	n/a
Sanitizer	n/a	n/a
Session	3/3	0
StrongUpdates	0/0	0
Sum	117/121	9

CONCLUSIONS

- FlowDroid - Novel and highly precise static analysis tool for Android apps.
- Accurately models Android lifecycle and callbacks.
- On-demand taint analysis algorithms allow for strong sensitivities with acceptable performance.
- DroidBench - Benchmark suite of Android apps for security benchmarking.