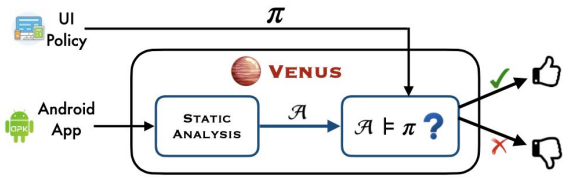
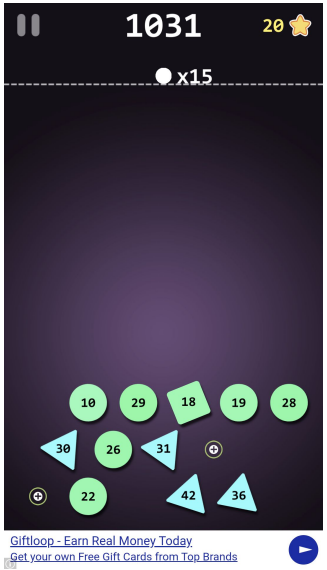
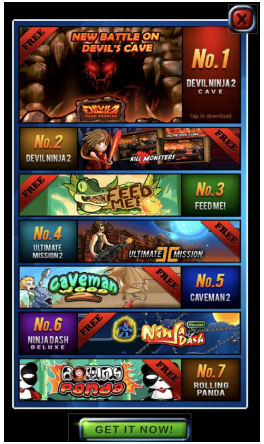
Checking Conformance against GUI Policies

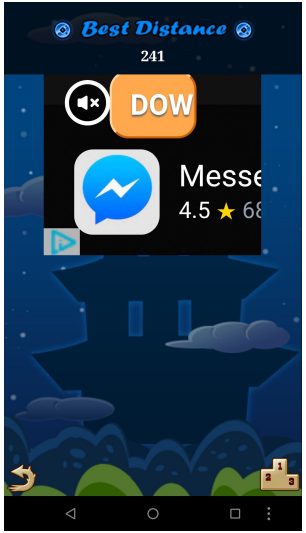
Zhen Zhang (UW), Yu Feng (UCSB), Michael D. Ernst (UW),
Sebastian Porst (Google), Isil Dillig (UT Austin)



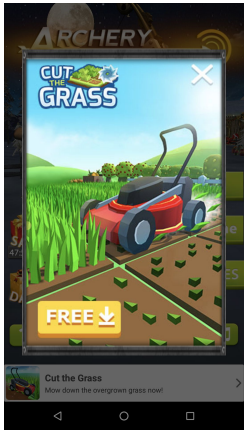
Virtually Presented at ESEC/FSE 2021

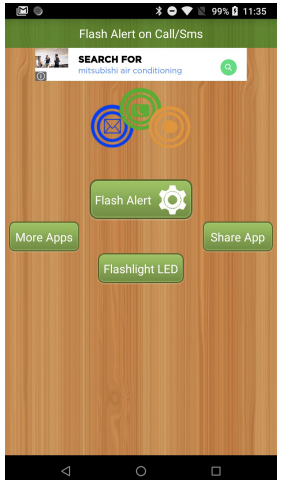


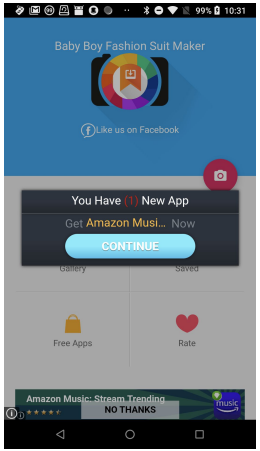




Motivations







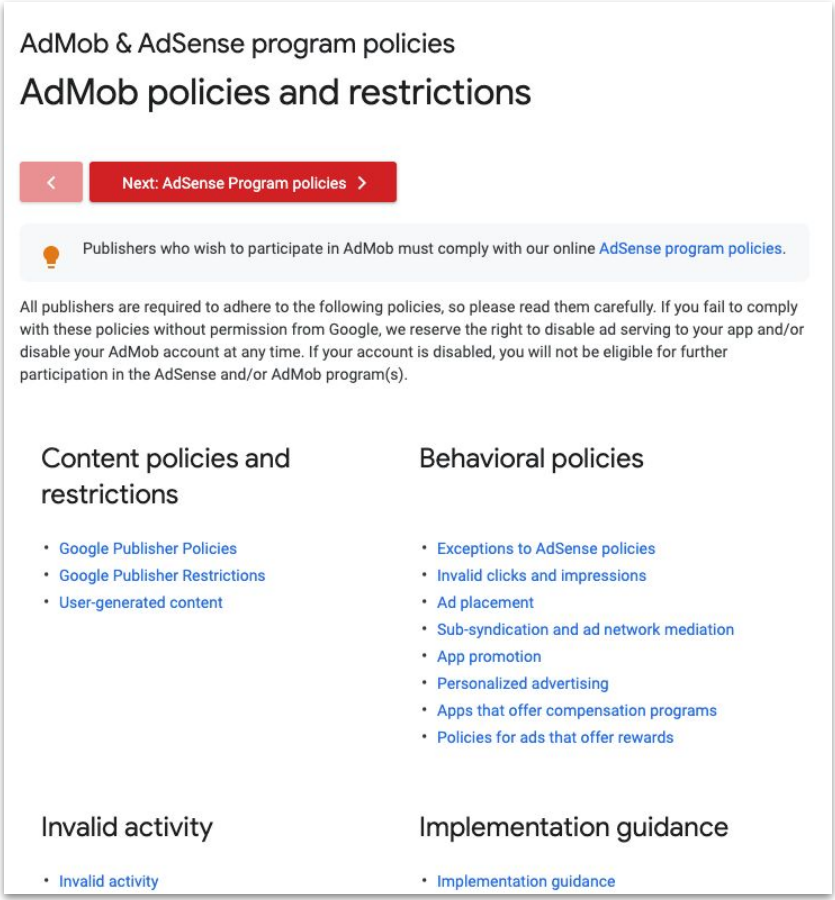
Motivations: What can be wrong with GUI in Apps?

In this work, we focus on the following types of problems:

- Ad Policy Violations
- Design Guideline Violations
- GDPR Regulation Violations

Ad Policy Violations

- Ad Platforms (e.g. AdMob) publish Ad Policies.
- Violations can harm *advertisers* and *users*.



AdMob & AdSense program policies

AdMob policies and restrictions

Next: AdSense Program policies

Publishers who wish to participate in AdMob must comply with our online AdSense program policies.

All publishers are required to adhere to the following policies, so please read them carefully. If you fail to comply with these policies without permission from Google, we reserve the right to disable ad serving to your app and/or disable your AdMob account at any time. If your account is disabled, you will not be eligible for further participation in the AdSense and/or AdMob program(s).

Content policies and restrictions

- Google Publisher Policies
- Google Publisher Restrictions
- User-generated content

Behavioral policies

- Exceptions to AdSense policies
- Invalid clicks and impressions
- Ad placement
- Sub-syndication and ad network mediation
- App promotion
- Personalized advertising
- Apps that offer compensation programs
- Policies for ads that offer rewards

Invalid activity

- Invalid activity

Implementation guidance

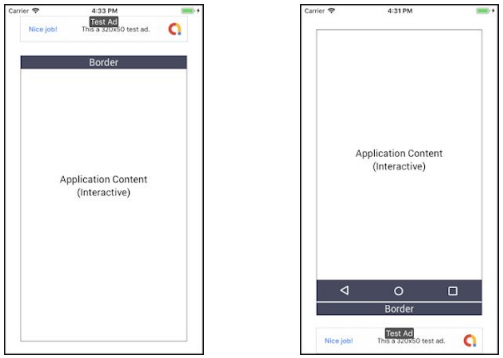
- Implementation guidance

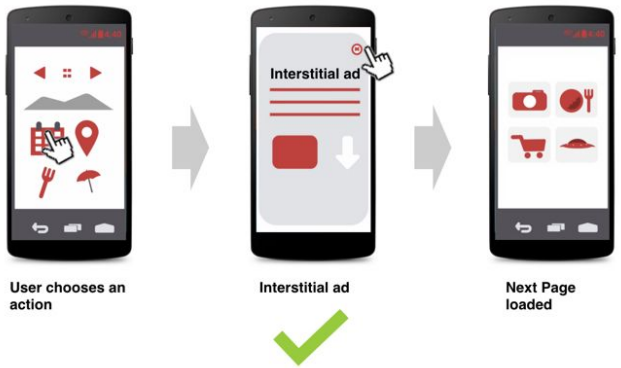
https://source.android.com/security/reports/Google_Android_Security_2018_Report_Final.pdf

Ad Policy Violations

Ad Policy examples:

- The size ratio between the ad and the screen is required to be greater than a minimum threshold.
- Full-screen ads should not overlap with other buttons.
- Ads should not be placed adjacent to a button.
- ….

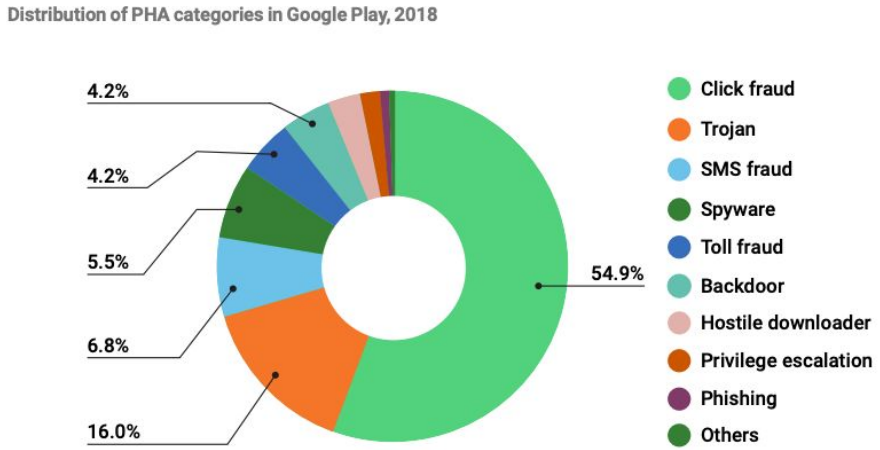




Ad Policy Violations

- Ad Platforms publish Ad Policies.
- Violations can harm *advertisers* and *users*.
- Problem is novel & severe
- Detection is ***hard***

Why: “Ad” is an abstract concept implemented with GUI elements



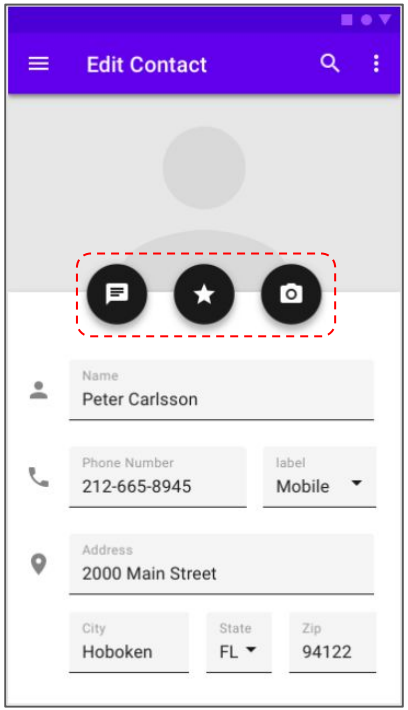
More than **50%** of potentially harmful apps in Google Play were *click fraud* (a major type of ad policy violation)

https://source.android.com/security/reports/Google_Android_Security_2018_Report_Final.pdf

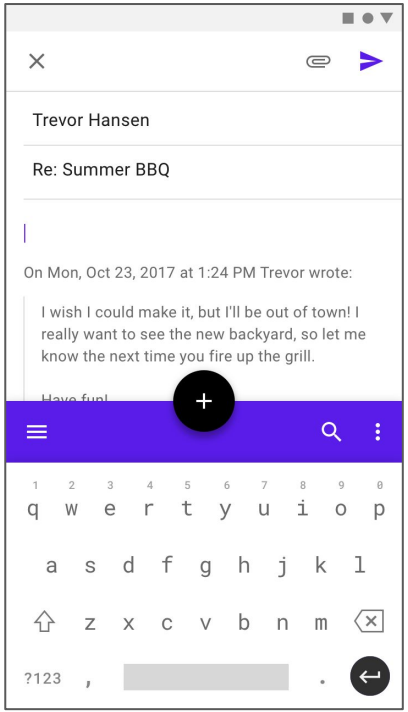
Design Guideline Violations

- App UI might violate the best practices, degrading UX (User Experiences)
- Examples:
  - Material Design
  - iOS Human Interface Guidelines
-

1. K. Moran, B. Li, C. Bernal-Ca´rdenas, D. Jelf, and D. Poshyvanyk, “Automated reporting of GUI design violations for mobile apps,” in *Proceedings of the 40th International Conference on Software Engineering*, 2018



Example 1: It is discouraged to put more than one FABs (Float Action Button) on the same screen.
https://material.io/components/buttons-floating-action-button#usage



Example 2: Don't attach a bottom app bar to the top of the keyboard.
https://material.io/components/app-bars-bottom#behavior

Design Guideline Violations

- App UI might violate the best practices, degrading UX (User Experiences)
- Examples:
  - Material Design
  - iOS Human Interface Guidelines
- Related work: *Automated reporting of GUI design violations for mobile apps (K. Moran et al., ICSE 2018)*
- Our work: first static checking against formal specifications

K. Moran, B. Li, C. Bernal-Ca´rdenas, D. Jelf, and D. Poshyvanyk, “Automated reporting of GUI design violations for mobile apps,” in *Proceedings of the 40th International Conference on Software Engineering*, 2018

General Data Protection Regulation (GDPR)

- Applications that display personalized ads should ***get user consent*** when they are started. (Android GDPR compliance guide[1])
- Personal data be processed lawfully, fairly and in a ***transparent*** manner (Article 5.1 of GDPR).
    - One concrete violation: La Liga “spy mode”[2].

Forbes

2,272 views | Jun 12, 2019, 09:20am EDT

La Liga Handed $280,000 GDPR Fine For 'Spying' On Fans Watching Pirated Streams

Steve McCaskill Senior Contributor

SportsMoney

I cover the intersection of sport and technology

1. https://developers.google.com/admob/android/eu-consent
2. https://techcrunch.com/2019/06/12/laliga-fined-280k-for-soccer-apps-privacy-violating-spy-mode

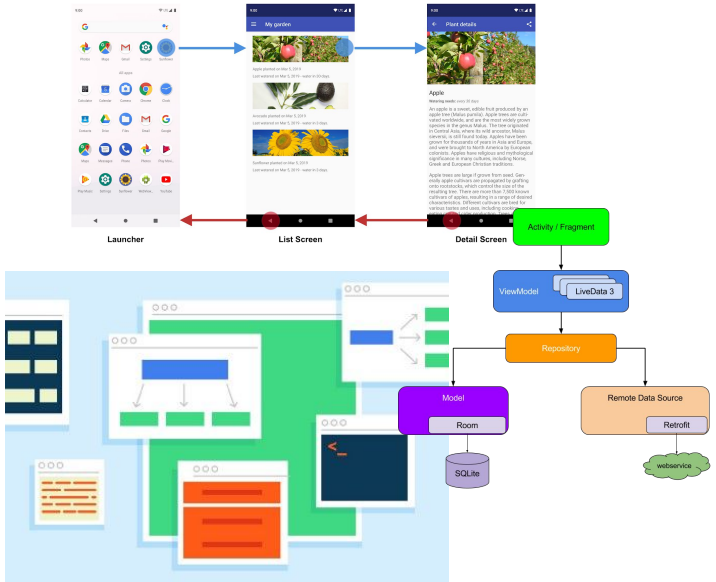
Background: Android GUI in a Nutshell

Next up: How are these violations **implemented**?

Ad Policy Violations

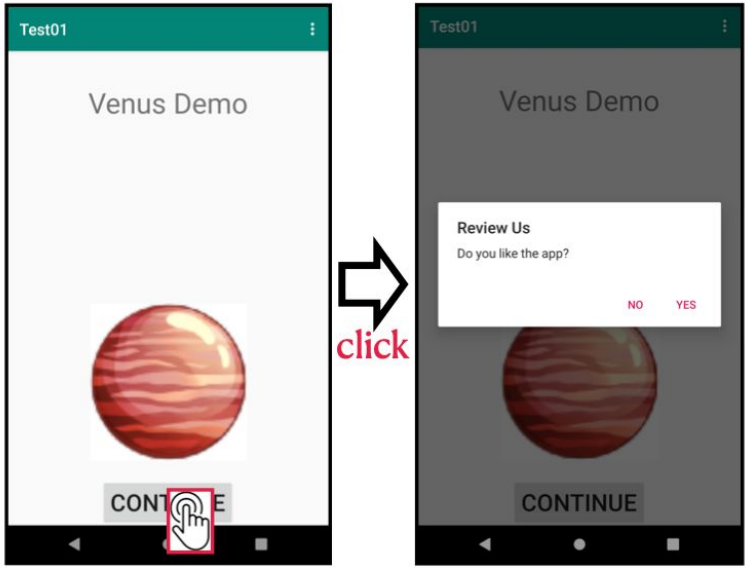
Design Guideline Violations

GDPR Regulation Violations



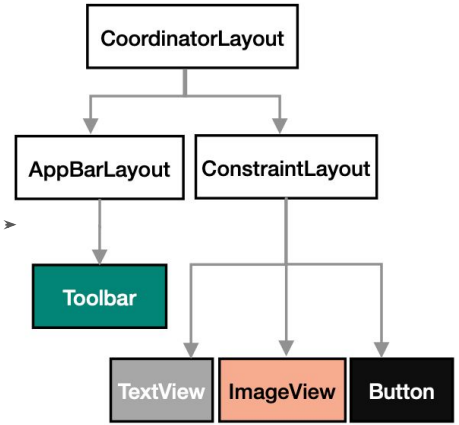
Background: Android GUI in a Nutshell

- GUI is central to mobile applications.
- Each app has many “Window”s (Activity/Dialog).
- Each “Window” has its lifecycles.
- One “Window” might transit to another “Window”.



Background: Android GUI in a Nutshell

- GUI is central to mobile applications.
- Each app has many “Window”s (Activity/Dialog).
- Each “Window” has its lifecycles.
- One “Window” might transit to another “Window”.
- Each window has a **hierarchy** of view nodes
  a. Declared in XML, or
  b. Constructed imperatively in code

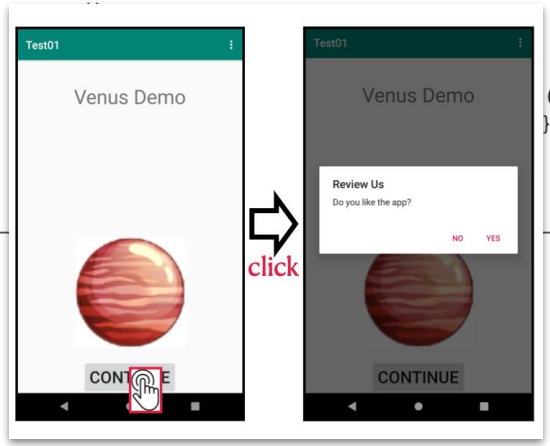



```
1 <ConstraintLayout>
2     ...
3     <TextView android:id="@+id/demo_title"
          android:text="Default title" ... />
4     <ImageView app:
          layout_constraintTop_toBottomOf="@+id/
          demo_title" ... />
5     <Button android:id="@+id/continue_button"
          android:text="CONTINUE" ... />
6 </ConstraintLayout>
```

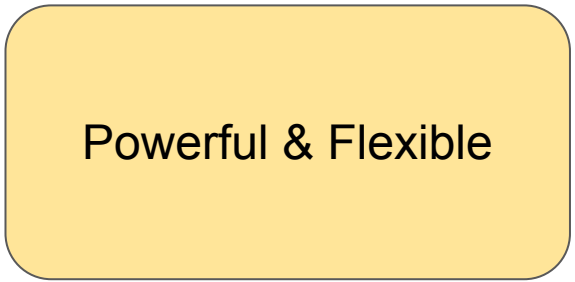
Background: Android GUI in a Nutshell

- GUI is central to mobile applications.
- Each app has many “Window”s (Activity/Dialog).
- Each “Window” has its lifecycles.
- One “Window” might transit to another “Window”.
- Each window has a **hierarchy** of view nodes
  - Declared in XML
  - Constructed imperatively
- View nodes might be registered with *event handlers*, e.g. `onClick`

```
1 class MainActivity extends Activity {
2   void onCreate(...) {
3     ...
4     setContentView(R.layout.activity_main);
5     TextView demoTitle = findViewById(R.id.
          demo_title);
6     demoTitle.setText("Venus Demo");
7     Button continueButton = findViewById(R.id.
          continue_button);
8     continueButton.setOnClickListener(new View.
          OnClickListener() {
9       void onClick(View v) {
10        AlertDialog d = new d.Builder(...).create
11                                                  () {
12                                                  }
13
14
15  } ...
```



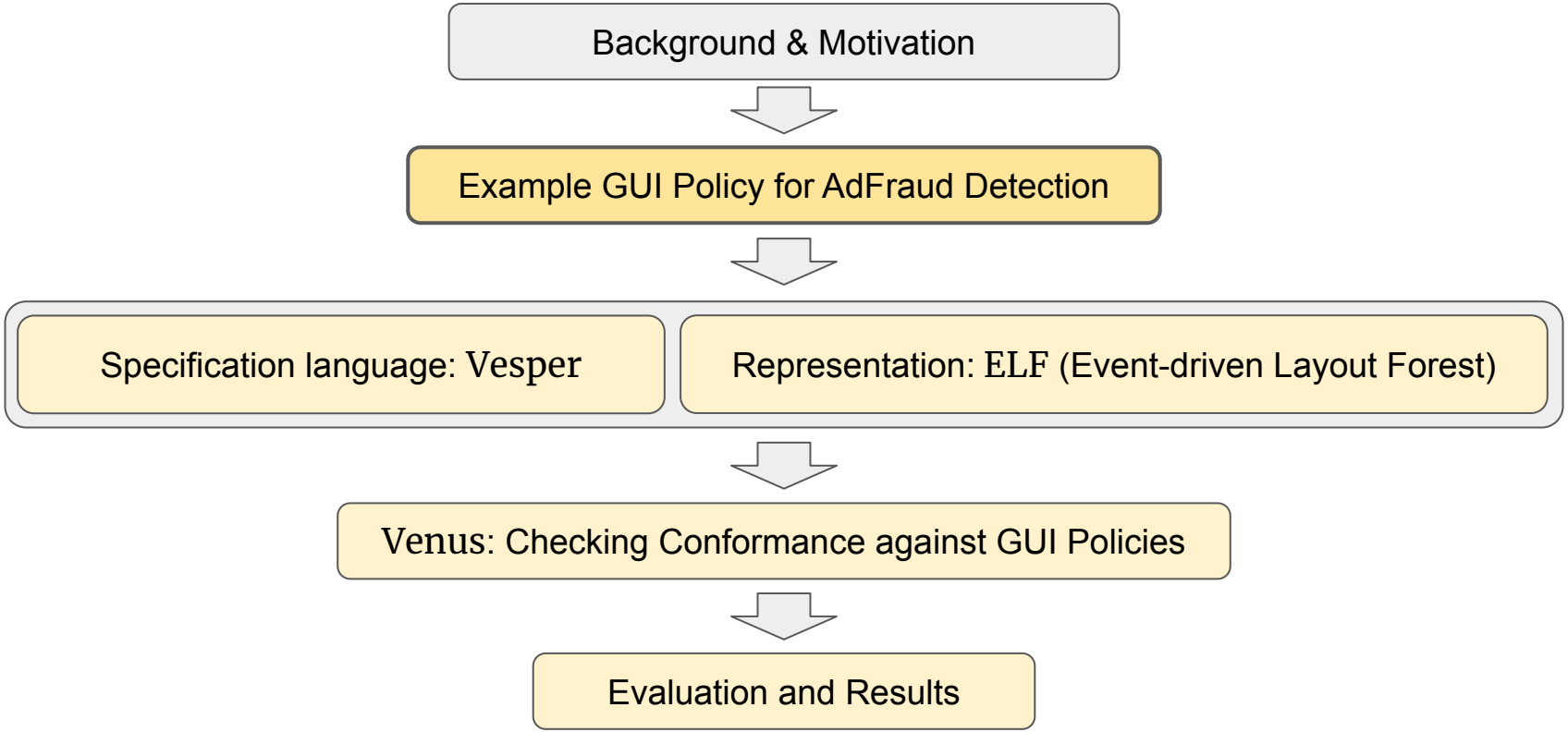
Background: Android GUI in a Nutshell



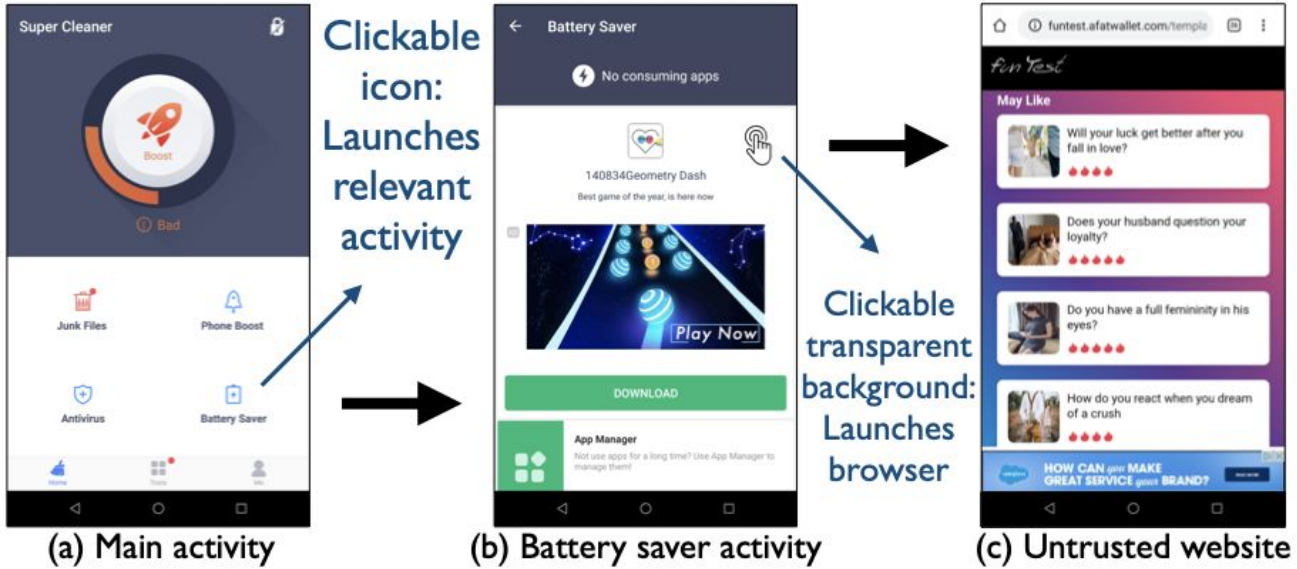
but

- **For attacker**: ability to create a wide variety of apps with GUI for harmful or fraudulent intentions
- **For developer**: possible to create GUIs with design defects which degrades user experience

Where we are now..



Example GUI Policy for AdFraud Detection



(a) Main activity (b) Battery saver activity (c) Untrusted website

Example GUI Policy for AdFraud Detection

Policy: Transparent background should not be clickable (and leads to unwanted contents, typically ads).

1. View bg
2. `assume` $(\exists v.(\text{View}(v) \wedge \text{background}(bg, v)))$
3. `let` $\text{popAd}(v) = \exists v'.\ (\text{showWindow}(v, \texttt{click}, v') \wedge \text{AdView}(v'))$
4. `assert` $(\text{transparent}(bg) \rightarrow \neg \text{popAd}(bg))$

bg is a background view associated some other view.

bg's attributes indicates its transparency.

When *bg* is clicked, it will lead to ads.

https://support.google.com/admanager/answer/7031536?hl=en

Recap: Challenges of Checking GUI Problems

Android GUI is powerful and flexible

Challenge #1: Reason about the semantics of GUI API, control flow, data-flow ...

No existing repository of clear and formal policies for Android GUI

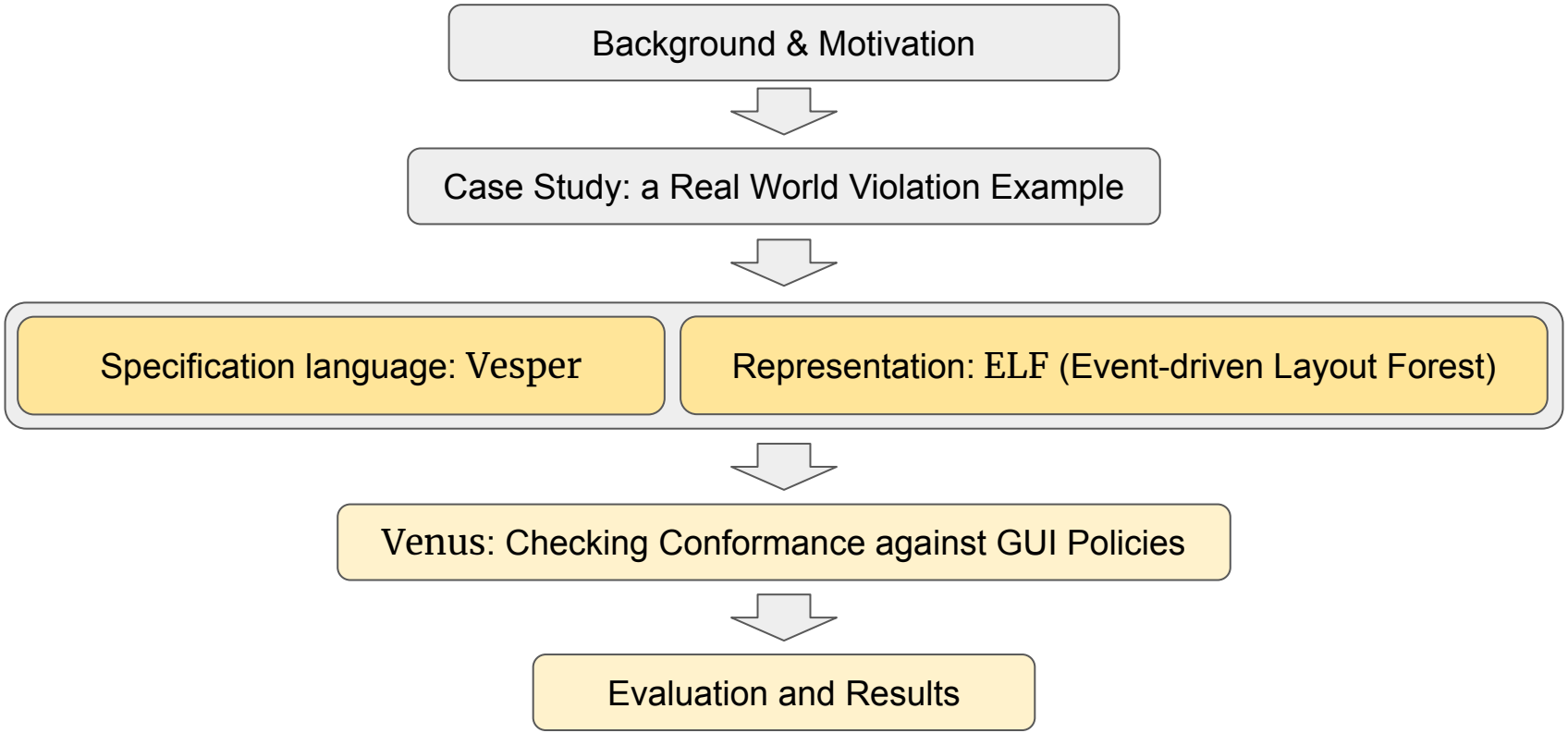
Challenge #2: Detection and analysis is ad-hoc, manual and not scalable

Contributions

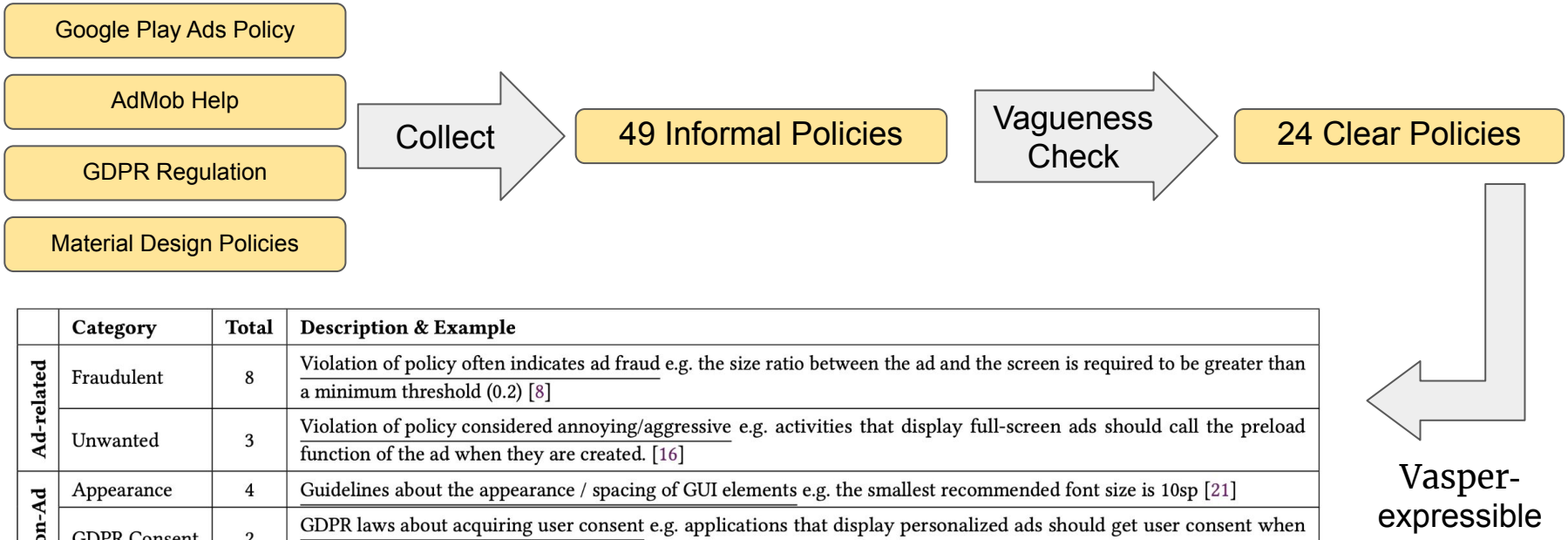
- **Vesper**: a formal specification language for describing GUI policies
- **Event-driven Layout Forest (ELF)**: a program abstraction and a static analysis technique for generating this abstraction
- **Venus**[1]: the first tool for statically checking conformance between Android apps and GUI specifications
- Evaluated on **3** datasets (**2361** Android applications) with **17** formalized policies, significant improvement compared to VirusTotal and FraudDroid

1. Open Source Code: https://github.com/izgzhen/ui-checker

Next...


Background & Motivation
Case Study: a Real World Violation Example
Specification language: Vesper
Representation: ELF (Event-driven Layout Forest)
Venus: Checking Conformance against GUI Policies
Evaluation and Results

GUI Policies Collection



	Category	Total	Description & Example
Ad-related	Fraudulent	8	Violation of policy often indicates ad fraud e.g. the size ratio between the ad and the screen is required to be greater than a minimum threshold (0.2) [8]
	Unwanted	3	Violation of policy considered annoying/aggressive e.g. activities that display full-screen ads should call the preload function of the ad when they are created. [16]
Non-Ad	Appearance	4	Guidelines about the appearance / spacing of GUI elements e.g. the smallest recommended font size is 10sp [21]
	GDPR Consent	2	GDPR laws about acquiring user consent e.g. applications that display personalized ads should get user consent when they are started [20]

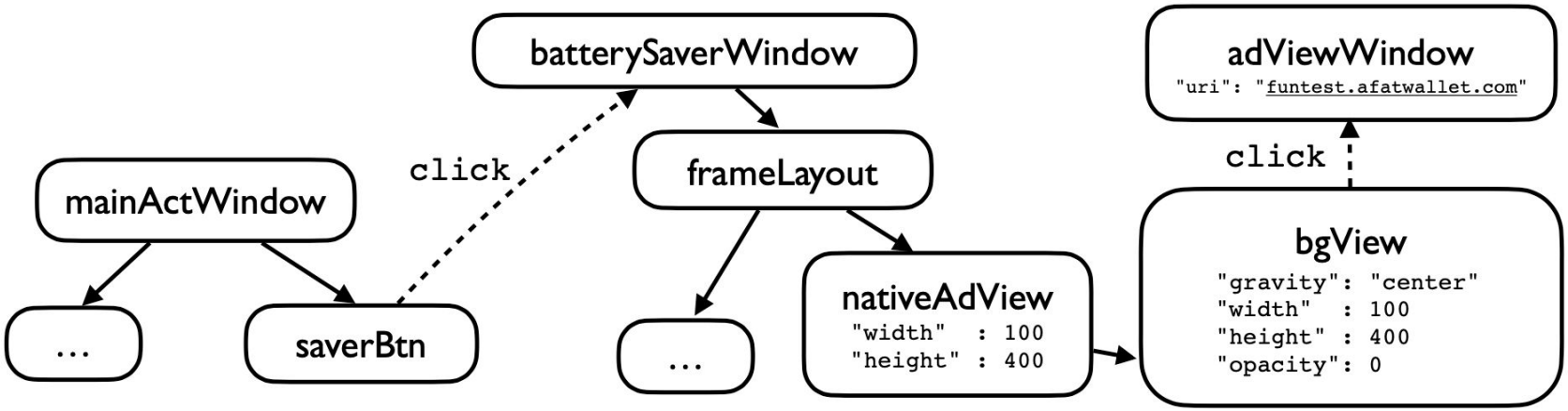
GUI Policies Formalization - Vesper language

$$
\begin{array}{rcl}
\text{Policy } \psi & \rightarrow & D;\ S;\ A \\
\text{Decl } D & \rightarrow & \tau\ v \mid D; D \\
\text{Stmt } S & \rightarrow & \texttt{let}\ v = \{v' \mid \Phi\} \\
 & & \mid \texttt{let}\ p(\overline{v}) = \Phi \mid \texttt{assume}\ \Phi \mid S; S \\
\text{Assert } A & \rightarrow & \texttt{assert}\ \Phi \mid A; A \\
\text{Expr } e & \rightarrow & v \mid \varepsilon \mid c \mid f(e_1, ..., e_n) \\
\text{Pred } \Phi & \rightarrow & p(\overline{e}) \mid \neg\Phi \mid \Phi_1 \vee \Phi_2 \\
 & & \mid \Phi_1 \wedge \Phi_2 \mid \Phi_1 \rightarrow \Phi_2 \mid \forall v.\ \Phi \mid \exists v.\ \Phi \\
\text{Event } \varepsilon & \rightarrow & \text{click} \mid \text{longClick} \mid \ \ldots\ \mid \text{touch} \\
\text{Type } \tau & \rightarrow & \text{View} \mid \text{Dialog} \mid \ldots \mid \text{Button} \\
a & \in & \text{Attributes}, \qquad c \in \text{Int} \quad f \in \text{Built-in fns} \\
p & \in & \text{Built-in predicates} \cup \text{User-defined predicates}
\end{array}
$$

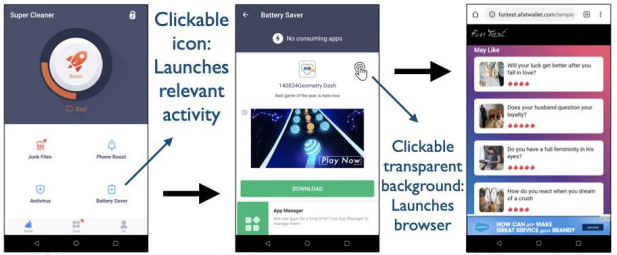
Built-in predicate examples:

- *Dialog(v)*: *v* is a dialog view
- *contains(u, v)*: *u* contains *v* as a sub-view
- *showWindow(u, e, v)*: Event *e* on *u* results immediately in display of element *v*

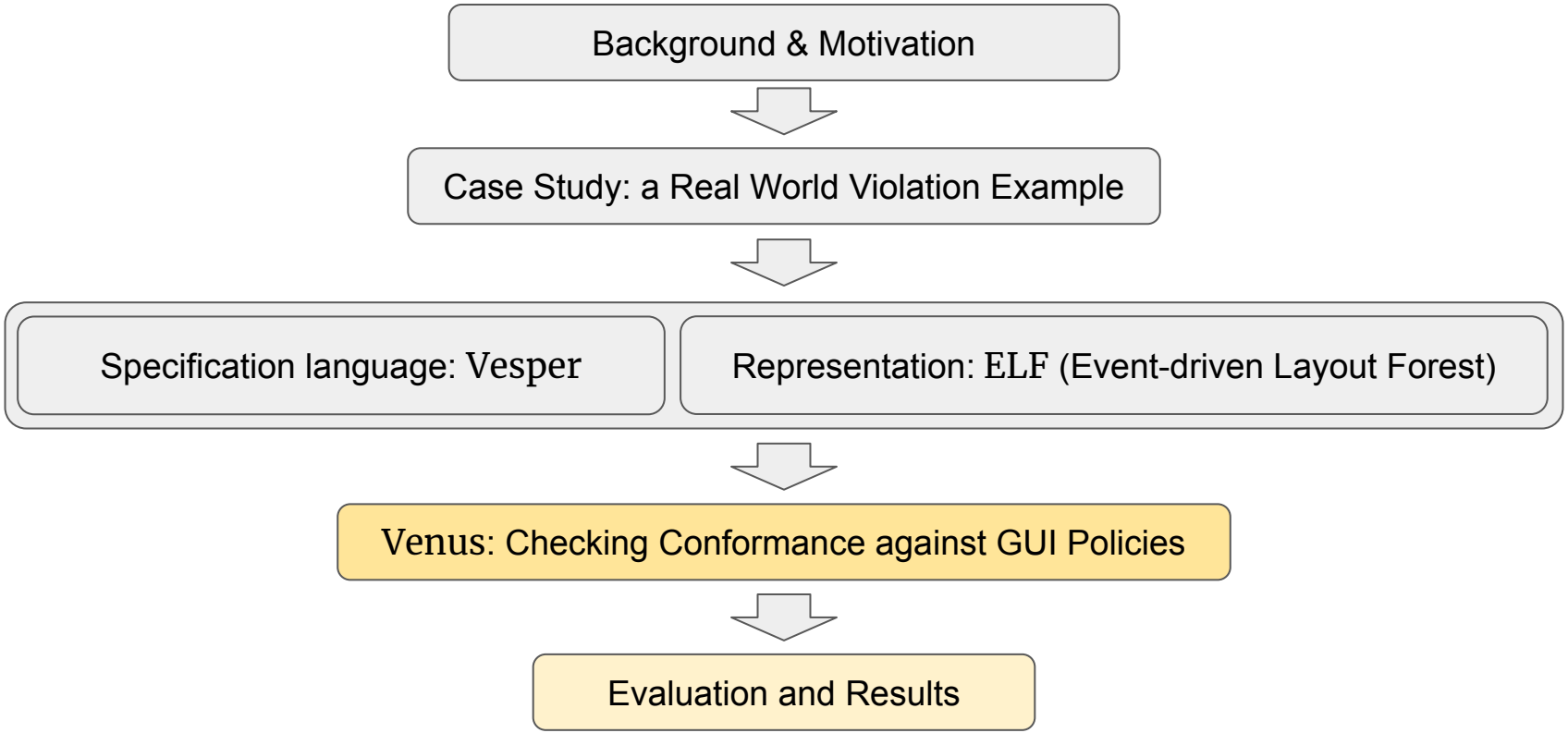
Event-driven Layout Forest (ELF)



- **Node**: GUI Element (*Window*, *View*, etc.)
- **Edge**: spatial (solid) / behavioral (dashed)
  - **Event**: `click`, `touch` etc.

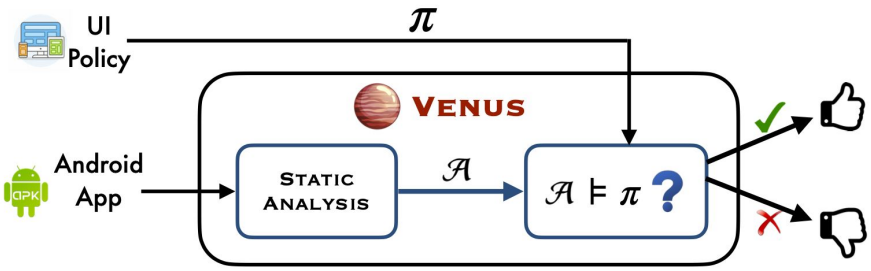


Next...


Background & Motivation
Case Study: a Real World Violation Example
Specification language: Vesper
Representation: ELF (Event-driven Layout Forest)
Venus: Checking Conformance against GUI Policies
Evaluation and Results

Venus: Checking Conformance against GUI Policies

- Core static analysis
  - XML Resource Analysis
  - Inter-procedural Data-flow Analysis
  - ELF generation
- Implementation
  - Built on top of Soot [1] framework and its SPARK framework for points-to analysis
  - Leverage IC3 tool [2] for Inter-Component Communication Analysis
  - Soufflé [3] for datalog solving



Why static analysis*?* (1) **Cheaper**: GUI interactions can lead to explosive number of concrete states (2) **Higher Coverage**: Malicious code can depend on dynamic triggers which is hard to satisfy.

1. https://github.com/soot-oss/soot
2. https://github.com/siis/ic3
3. https://souffle-lang.github.io/

Evaluation: Datasets

Dataset Name	Description
Google Play	We collected 1488 popular applications that were available on the Google Play Store in Jan 2019.
GPP	Applications flagged as potential malware by Google's internal tools and manually audited by Google security analysts (2019)
AdFraudBench	Benchmark collected in the FraudDroid paper (2018) [1]

1. Feng Dong, Haoyu Wang, Li Li, Yao Guo, Tegawendé F. Bissyandé, Tianming Liu, Guoai Xu, and Jacques Klein. 2018. FraudDroid: automated ad fraud detection for Android apps. In Proceedings of the 2018 26th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering (ESEC/FSE 2018).

Evaluation: Venus Summary

Dataset	# apps	# violating apps	# violations	Recall	Precision	Avg. time (s)
Google Play	1488	711	1645	N/A	**89.2%**	465.3
GPP	773	243	391	**86.8%**	**94.7%**	464.7
AdFraudBench	100	54	90	**91.2%**	**96.3%**	302.1
All	2361	1008	2126	N/A	**91.3%**	458.2

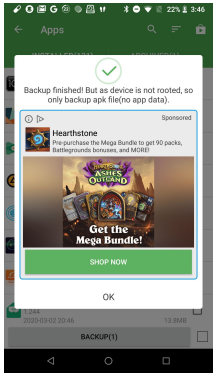
Evaluation: Venus Summary

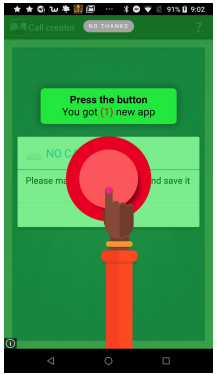
About ½ the time is spent on pre-analysis, and the other ½ for ELF analysis. Datalog solving cost is negligible.

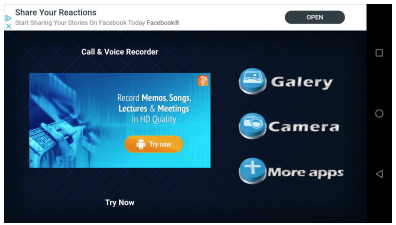
Dataset	# apps	# violating apps	# violations	Recall	Precision	Avg. time (s)
Google Play	1488	711	1645	N/A	**89.2%**	465.3
GPP	773	243	391	**86.8%**	**94.7%**	464.7
AdFraudBench	100	54	90	**91.2%**	**96.3%**	302.1
All	2361	1008	2126	N/A	**91.3%**	458.2

Results I: Google Play Dataset

- Identified **11** APKs with *previously unknown* Ad Policy violations
  - e.g. *Ads should not be placed in a location that covers up or hides any area that users have interest in viewing during typical interaction.*
- Identified **24** APKs with Design Guidelines violations
  - e.g. *Smallest recommended main text font size in Material design is 10sp.*
- Identified **16** APKs with GDPR violations
  - e.g. *Access personal information without displaying a consent form*

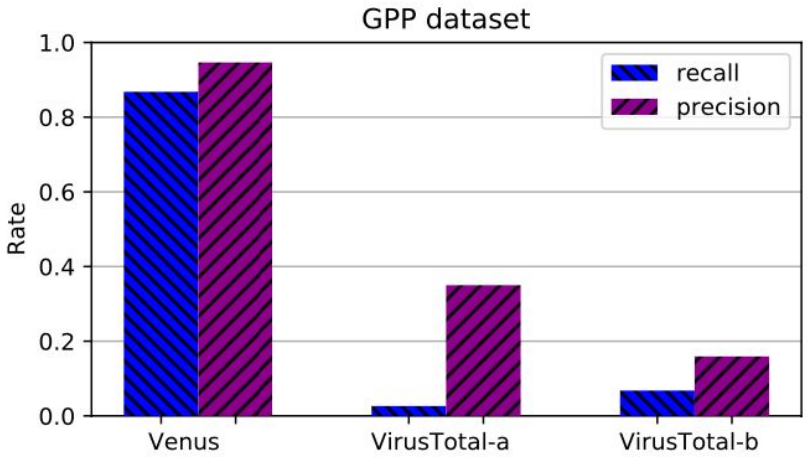






Results II: GPP Dataset

- **Compared to** VirusTotal [1]
  - **2.7x** improvement in precision
  - **12.8x** improvement in recall
- **Source of errors**
  - **False negatives:** foreign binary code
  - **False positives:** imprecision in pointer analysis



1. https://www.virustotal.com/: a service that aggregates many commercial antivirus/scanning engines

Results III: AdFraudBench Dataset

	Venus	FraudDroid [1]	VirusTotal-a	VirusTotal-b
Precision	**96.3%**	91.8%	79.6%	75.0%
Recall	**91.2%**	78.9%	75.4%	89.5%

Significantly better recall

1. Feng Dong, Haoyu Wang, Li Li, Yao Guo, Tegawendé F. Bissyandé, Tianming Liu, Guoai Xu, and Jacques Klein. 2018. FraudDroid: automated ad fraud detection for Android apps. In Proceedings of the 2018 26th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering (ESEC/FSE 2018).

Recap: Contributions

- **Venus**[1]: the first tool for statically checking conformance between Android apps and GUI specifications
- **Vesper**: a formal specification language for describing GUI policies
- **Event-driven Layout Forest (ELF)**: a program abstraction and a static analysis technique for generating this abstraction
- **Evaluated** on **3** datasets (**2361** Android applications) with **17** formalized policies, significant improvement compared to VirusTotal and FraudDroid

Thanks!

The work is partially supported by the funds:

- NSF Grants #1908494, #1908304
- CCF-#2005889, CNS-#1822251
- Google Faculty Research Award

More resources:

- Code: https://github.com/izgzhen/ui-checker
- Draft: https://bit.ly/fse21-venus-preprint

Supplementary Materials

Predicates

Element type predicates

Button(v), Dialog(v), ImageView(v), AdView(v) …

Spatial predicates:

height(v, h)	View v has height h
width(v, w)	View v has width w
textSize(v, s)	Text view v has text size s
transparent(v)	View v is transparent
contains(u, v)	u contains v as a sub-view
background(u, v)	u is the background container of v

Behavioral predicates:

entryView(v)	v is the top-level window that is displayed when the app starts
invoke(u, e, m)	User event e on GUI element u directly causes invocation of method m
showWindow(u, e, v)	Event e on u results immediately in display of element v
launchDialog(w, e, v)	Window w's event e causes new dialog v to be immediately displayed
startMarketplace(e, v)	Event e on v results immediately in starting a new marketplace window
startBrowser(e, v)	Event e on v results immediately in starting a new browser window

Source code predicates	
loadView(l, m, v, N)	load layout N to v at location l of method m
addView(l, m, v_1, v_2)	v_2 is added as sub-view of v_1 at l in method m
setContentView(l, m, v_1, v_2)	add v_2 as content view of v_1 at l in method m
setAttrib(l, m, v, a, v')	attribute a of v is set to v' at l in method m
set$X$Listener(l, m, v, m')	Method m' is set as v's X listener
showWindow(l, m, v)	Location l has a call to display window v
icc$(l, m, intent)$	Perform ICC using *intent* at l of method m
mainAct(A)	A is the app's main activity
Pre-analysis predicates	
inCtx(m, c)	c is a calling context of method m
aval(c, l, v, a)	v has abstract value a at location l in context c
pointsTo(c, l, v, o)	v points to object o at location l in context c
pointsTo(c, l, o, f, o')	The f field of o points to o' at l in context c
call$^*(c, m, m')$	m directly or transitively calls m' in context c
hasType(o, τ)	Heap object o has type τ
…	…
Output predicates	
node(o, τ)	o is a GUI element node of type τ in Elf
sAttrib(o, a_s, val)	node o has spatial attribute a_s with value val
bAttrib(o, a_b, val)	node o has behavioral attribute a_b with val
entryView(v)	v is a window shown on app startup
sEdge(o, o')	view o contains view o'
bEdge(o, ε, o')	view o leads to view o' under event ε
rootView(o_1, o_2)	o_1 has root view o_2

Inference Rules

$$\mathrm{node}(o,\tau) \Leftarrow \mathrm{pointsTo}(\_,\_,v,o),\ \mathrm{hasType}(o,\tau),\ \tau <: \mathrm{View}. \quad (1)$$

$$\mathrm{rootView}(o,o') \Leftarrow \mathrm{setContentView}(l,m,v,v'),\ \mathrm{inCtx}(m,c),\ \mathrm{pointsTo}(c,l,v,o),\ \mathrm{pointsTo}(c,l,v',o'). \quad (2)$$

$$\mathrm{entryView}(o) \Leftarrow \mathrm{mainAct}(A),\ \mathrm{instanceOf}(o,A),\ \mathrm{rootView}(o',o). \quad (3)$$

$$\mathrm{bAttrib}(o,X,m) \Leftarrow \mathrm{node}(o,\_),\ \mathrm{setXListener}(l,v,m),\ \mathrm{inCtx}(m,c),\ \mathrm{pointsTo}(c,l,v,o). \quad (4)$$

$$\mathrm{sAttrib}(o,a,\bot) \Leftarrow \mathrm{node}(o,\mathrm{View}),\ a \in \mathrm{Attribs}(\Psi). \quad (5)$$

$$\mathrm{sAttrib}(o,a,val') \Leftarrow \mathrm{loadView}(l,m,v,N),\ \mathrm{inCtx}(m,c),\ \mathrm{pointsTo}(c,l,v,o).\ a \in \mathrm{Dom}(\Psi(N)),\ a \neq \mathrm{subview},\ \Psi(N)(a) = (T, val_0),\ \mathrm{sAttrib}(o,a,val),\ val' = val \sqcup \alpha(val_0) \quad (6)$$

$$\mathrm{sAttrib}(o,a,val'') \Leftarrow \mathrm{setAttrib}(l,m,v,a,v'),\ \mathrm{inCtx}(m,c),\ \mathrm{pointsTo}(c,l,v,o),\ \mathrm{aval}(c,l,v',val'),\ \mathrm{sAttrib}(o,a,val),\ val'' = val \sqcup \alpha(val'). \quad (7)$$

$$\mathrm{sEdge}(o,o') \Leftarrow \mathrm{loadView}(l,m,v,N),\ \mathrm{inCtx}(m,c),\ \mathrm{pointsTo}(c,l,v,o),\ o' \in \Psi(N)(\mathrm{subview}). \quad (8)$$

$$\mathrm{sEdge}(o_1,o_2) \Leftarrow \mathrm{addView}(l,m,v_1,v_2),\ \mathrm{inCtx}(m,c),\ \mathrm{pointsTo}(c,l,v_1,o_1),\ \mathrm{pointsTo}(c,l,v_2,o_2). \quad (9)$$

$$\mathrm{bEdge}(o_1,X,o_2) \Leftarrow \mathrm{bAttrib}(o_1,X,m),\ \mathrm{inCtx}(m,c),\ \mathrm{call}^*(c,m,m'),\ \mathrm{inCtx}(m',c'),\ \mathrm{showWindow}(l,m',v),\ \mathrm{pointsTo}(c',l,v,o_2). \quad (10)$$

$$\mathrm{bEdge}(o_1,X,o_2) \Leftarrow \mathrm{bAttrib}(o_1,X,m),\ \mathrm{inCtx}(m,c'),\ \mathrm{call}^*(c',m,m'),\ \mathrm{inCtx}(m',c),\ \mathrm{icc}(l,m',i),\ \mathrm{pointsTo}(c,l,i,o)\ \mathrm{pointsTo}(c,l,o,\text{“tgt”},o'),\ \mathrm{rootView}(o',o_2). \quad (11)$$