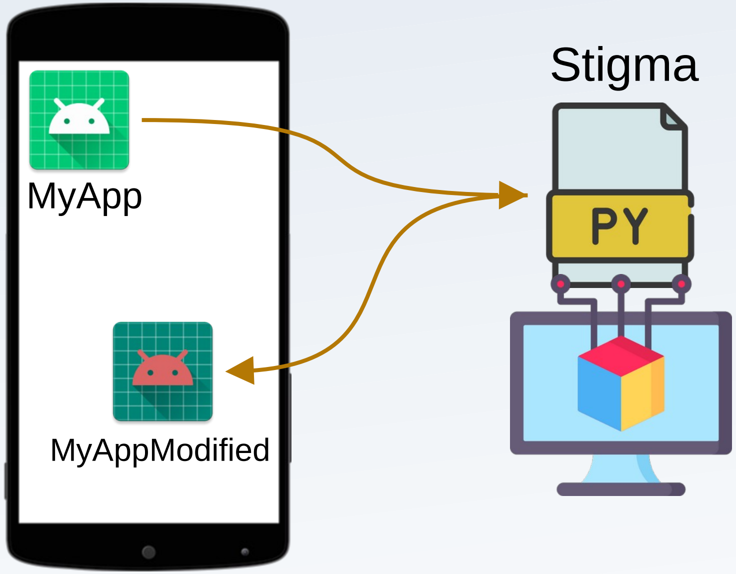
Stigma: A Tool for Modifying Closed-Source Android Applications



Ed Novak, Shaamyl Anwar, Saad Mahboob,
Shokhinabonu Tojieva, and Chelsea Rao

FRANKLIN & MARSHALL COLLEGE

Premise

- Android applications are (mostly) closed-source
  - Propriety, closed code-bases carefully guarded by large corp.
  - Elaborate obfuscation (ProGuard)

- Android has a permission system, but it is limited
  - Internet permission is very broad
  - Most apps require many permissions
    - → User fatigue and apathy

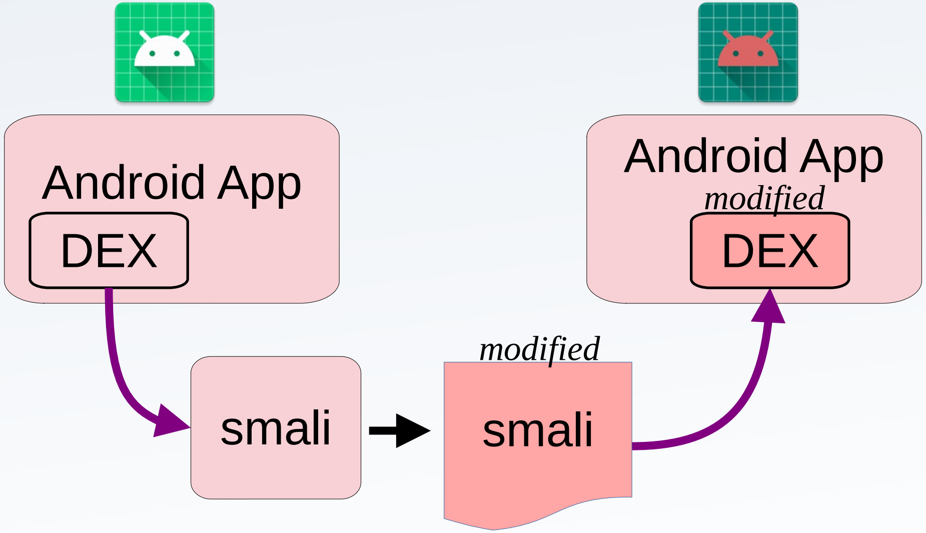
- Users often desire small or medium changes in popular apps
  - Is it tracking me? Can I stop it from doing that?
  - Can I remove ads?
  - Can I fix a bug, change the UI, or add a feature?
  - Does this app exhibit low-security functionality?

Main Idea and Background

- Modify closed-source Android applications
  - Maintain all app functionality
  - Allow user some partially automatic mechanism to specify desired modifications

- Run modified apps on real hardware

- Background
  - Android apps are distributed as DEX bytecode
    - Machine Code (binary assembly) for JVM (dalvik)
  - We can convert DEX bytecode to `smali`
    - Using a third party tool: apktool
    - Assembly code (human readable)
  - We can modify the smali bytecode and recompile to DEX



Smali Assembly Code

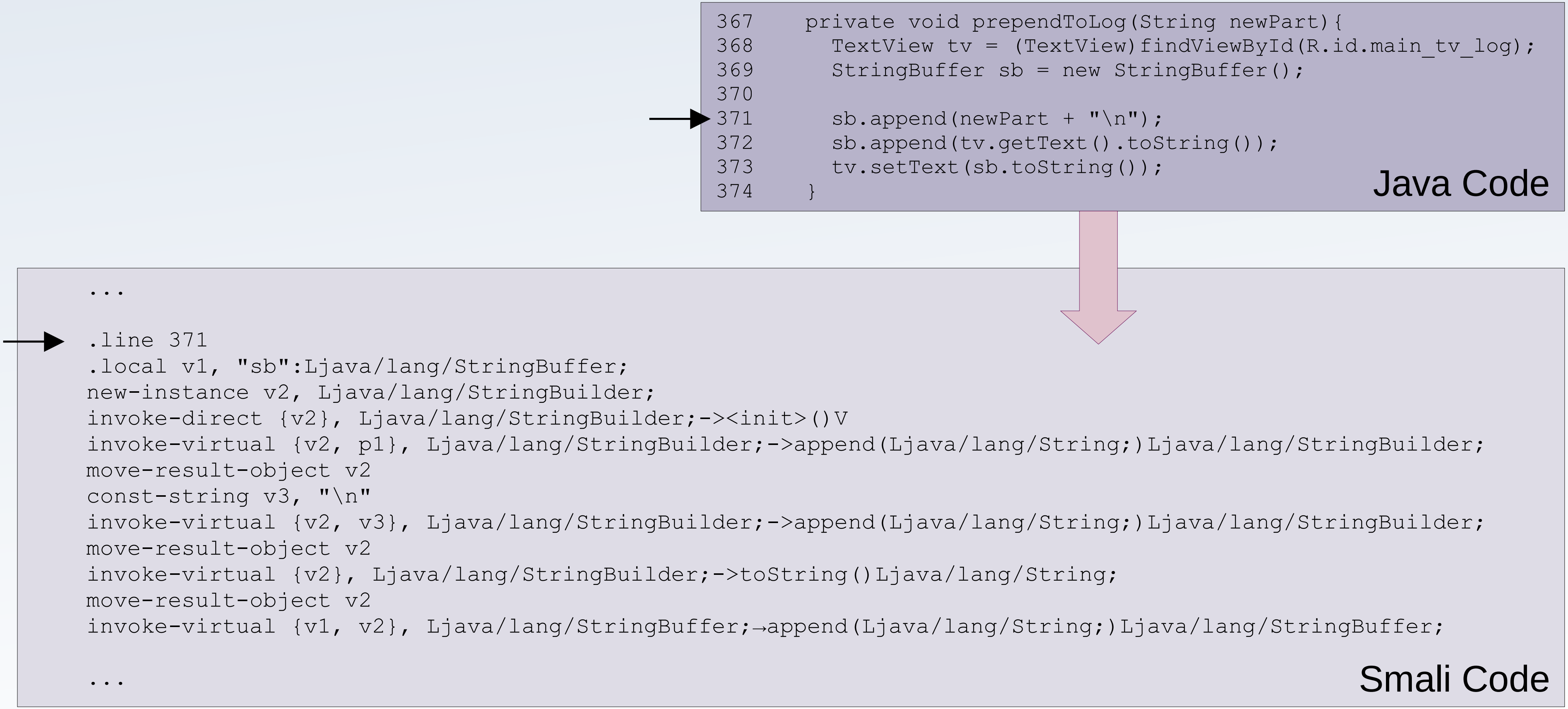
```
367    private void prependToLog(String newPart){
368      TextView tv = (TextView)findViewById(R.id.main_tv_log);
369      StringBuffer sb = new StringBuffer();
370
371      sb.append(newPart + "\n");
372      sb.append(tv.getText().toString());
373      tv.setText(sb.toString());
374    }
```

Java Code

```
.method private prependToLog(Ljava/lang/String;)V
    .locals 4
    .param p1, "newPart"    # Ljava/lang/String;

    .line 368
    const v0, 0x7f070059

    invoke-virtual {p0, v0}, Ledu/fandm/enovak/leaks/Main;->findViewById(I)Landroid/view/View;
    move-result-object v0
    check-cast v0, Landroid/widget/TextView;

    ...
```

Smali Code

Smali Assembly Code



```
367    private void prependToLog(String newPart){
368      TextView tv = (TextView)findViewById(R.id.main_tv_log);
369      StringBuffer sb = new StringBuffer();
370
371      sb.append(newPart + "\n");
372      sb.append(tv.getText().toString());
373      tv.setText(sb.toString());
374    }
```

Java Code

```
...

.line 371
.local v1, "sb":Ljava/lang/StringBuffer;
new-instance v2, Ljava/lang/StringBuilder;
invoke-direct {v2}, Ljava/lang/StringBuilder;-><init>()V
invoke-virtual {v2, p1}, Ljava/lang/StringBuilder;->append(Ljava/lang/String;)Ljava/lang/StringBuilder;
move-result-object v2
const-string v3, "\n"
invoke-virtual {v2, v3}, Ljava/lang/StringBuilder;->append(Ljava/lang/String;)Ljava/lang/StringBuilder;
move-result-object v2
invoke-virtual {v2}, Ljava/lang/StringBuilder;->toString()Ljava/lang/String;
move-result-object v2
invoke-virtual {v1, v2}, Ljava/lang/StringBuffer;→append(Ljava/lang/String;)Ljava/lang/StringBuffer;

...
```

Smali Code

Smali Assembly Code

Smali is the most convenient and intuitive form of the code that we have!

```
367    private void prependToLog(String newPart){
368      TextView tv = (TextView)findViewById(R.id.main_tv_log);
369      StringBuffer sb = new StringBuffer();
370
371      sb.append(newPart + "\n");
372      sb.append(tv.getText().toString());
373      tv.setText(sb.toString());
374    }
```

Java Code

```
...

.line 371
.local v1, "sb":Ljava/lang/StringBuffer;
new-instance v2, Ljava/lang/StringBuilder;
invoke-direct {v2}, Ljava/lang/StringBuilder;-><init>()V
invoke-virtual {v2, p1}, Ljava/lang/StringBuilder;->append(Ljava/lang/String;)Ljava/lang/StringBuilder;
move-result-object v2
const-string v3, "\n"
invoke-virtual {v2, v3}, Ljava/lang/StringBuilder;->append(Ljava/lang/String;)Ljava/lang/StringBuilder;
move-result-object v2
invoke-virtual {v2}, Ljava/lang/StringBuilder;->toString()Ljava/lang/String;
move-result-object v2
invoke-virtual {v1, v2}, Ljava/lang/StringBuffer;→append(Ljava/lang/String;)Ljava/lang/StringBuffer;

...
```

Smali Code

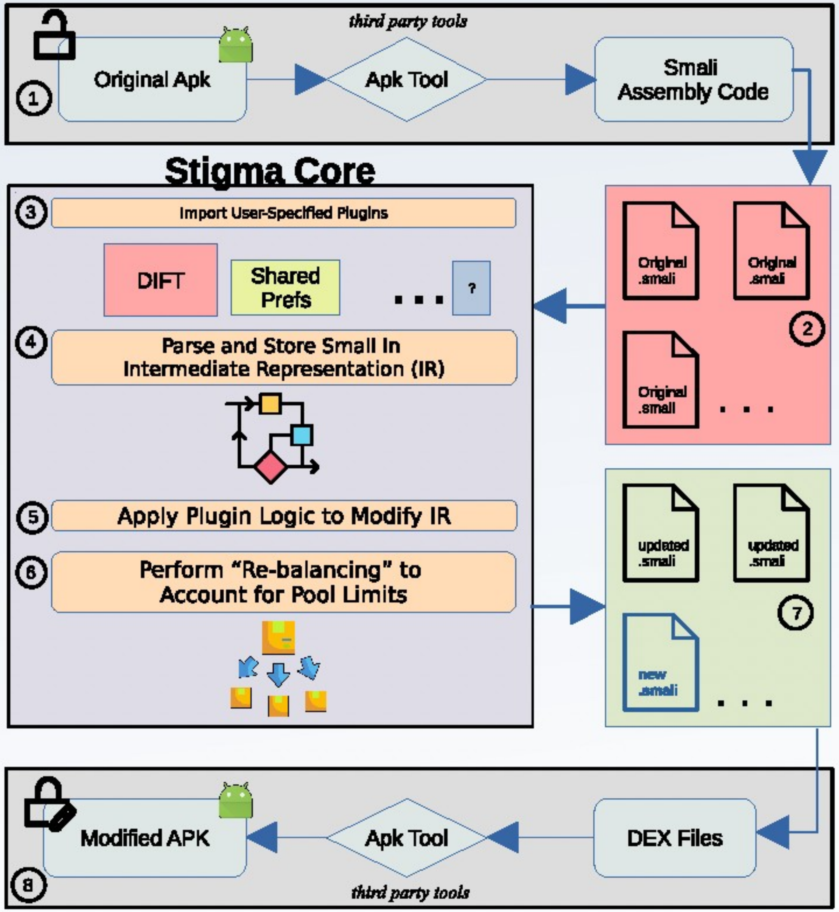
```
.method private prependToLog(Ljava/lang/String;)V
    .locals 4
    .param p1, "newPart"    # Ljava/lang/String;

    .line 368
    const v0, 0x7f070059

    invoke-virtual {p0, v0}, Ledu/fandm/enovak/leaks/Main;->findViewById(I)Landroid/view/View;
    move-result-object v0
    check-cast v0, Landroid/widget/TextView;

    .line 369
    .local v0, "tv":Landroid/widget/TextView;
    new-instance v1, Ljava/lang/StringBuffer;
    invoke-direct {v1}, Ljava/lang/StringBuffer;-><init>()V

    .line 371
    .local v1, "sb":Ljava/lang/StringBuffer;
    new-instance v2, Ljava/lang/StringBuilder;
    invoke-direct {v2}, Ljava/lang/StringBuilder;-><init>()V
    invoke-virtual {v2, p1}, Ljava/lang/StringBuilder;->append(Ljava/lang/String;)Ljava/lang/StringBuilder;
    move-result-object v2
    const-string v3, "\n"
    invoke-virtual {v2, v3}, Ljava/lang/StringBuilder;->append(Ljava/lang/String;)Ljava/lang/StringBuilder;
    move-result-object v2
    invoke-virtual {v2}, Ljava/lang/StringBuilder;->toString()Ljava/lang/String;
    move-result-object v2
    invoke-virtual {v1, v2}, Ljava/lang/StringBuffer;→append(Ljava/lang/String;)Ljava/lang/StringBuffer;

    .line 372
    invoke-virtual {v0}, Landroid/widget/TextView;->getText()Ljava/lang/CharSequence;
    move-result-object v2
    invoke-interface {v2}, Ljava/lang/CharSequence;->toString()Ljava/lang/String;
    move-result-object v2
    invoke-virtual {v1, v2}, Ljava/lang/StringBuffer;->append(Ljava/lang/String;)Ljava/lang/StringBuffer;

    .line 373
    invoke-virtual {v1}, Ljava/lang/StringBuffer;->toString()Ljava/lang/String;
    move-result-object v2
    invoke-virtual {v0, v2}, Landroid/widget/TextView;->setText(Ljava/lang/CharSequence;)V

    .line 374
    return-void
.end method
```

Smali Code

```
367   private void prependToLog(String newPart){
368     TextView tv = (TextView)findViewById(R.id.main_tv_log);
369     StringBuffer sb = new StringBuffer();
370
371     sb.append(newPart + "\n");
372     sb.append(tv.getText().toString());
373     tv.setText(sb.toString());
374   }
```

Java Code

Stigma Pipeline



Plugins

Existing App Code

```
...

.line 371
.local v1, "sb":Ljava/lang/StringBuffer;
new-instance v2, Ljava/lang/StringBuilder;
invoke-direct {v2}, bar
invoke-virtual {v2, p1}, foo
move-result-object v2
const-string v3, "\n"
invoke-virtual {v2, v3}, foo
move-result-object v2
invoke-virtual {v2}, bar
move-result-object v2
invoke-virtual {v1, v2}, biz

...
```

New Code to Inject

```
const-string v6, “New Line Used!”
invoke-direct Ljava/lang/System->out->println(Ljava/lang/String;)V
```

- Framework for users to define their modifications.
  - Core concept: user specifies “hooks” where Stigma inserts new smali code into the app
  - “Hooks” can be…
    - specific smali instructions (`move`, `add-int`, `if-eq`, `iget`, `invoke-*`, etc.)
    - conceptual code locations (the start of the `onCreate()` method in launcher activities, (the start of every method call, etc.)
  - The user specifies smali instructions to insert at that point
    - Customizable number of “free” registers
    - An “in-stigma-memory” object representing the method and class where the new code is inserted
  - Stigma maintains python classes to represent all smali instructions, methods, and classes (files).

- We’ve defined two plugins already to demonstrate and evaluate the Stigma system
  - Dynamic Information Flow Tracking (DIFT) plugin
  - SharedPreferences plugin

Plugin Details

Dynamic Information Flow Tracking (DIFT)

- Why? To track clandestine use of sensitive information
- Functionality
  - (1) **Originate:** Identify API methods and other sources of sensitive information
  - (2) **Tag & Propagate:** Mark variables (registers) that contain sensitive info with “tags.” Our plugin stores the the tags in static class fields in new classes added to the app’s code-base
  - (3) **Terminate:** Identify API methods that consume, transmit, or store data. Check the input parameters for tags when they’re called. Alert the user and/or log activity.
- Just a prototype that does steps one, two and three in a limited capacity for GPS data.

SharedPreferences Plugin

- Why? To identify malicious or negligent use of SharedPreferences API to store sensitive information
- Functionality
  - Print contents of (default) shared preferences database on app launch
- Features / Benefits
  - No root required
  - Capture real values during runtime
- Just a prototype that requires follow-up analysis of the database items printed / logged.

Technical Challenges

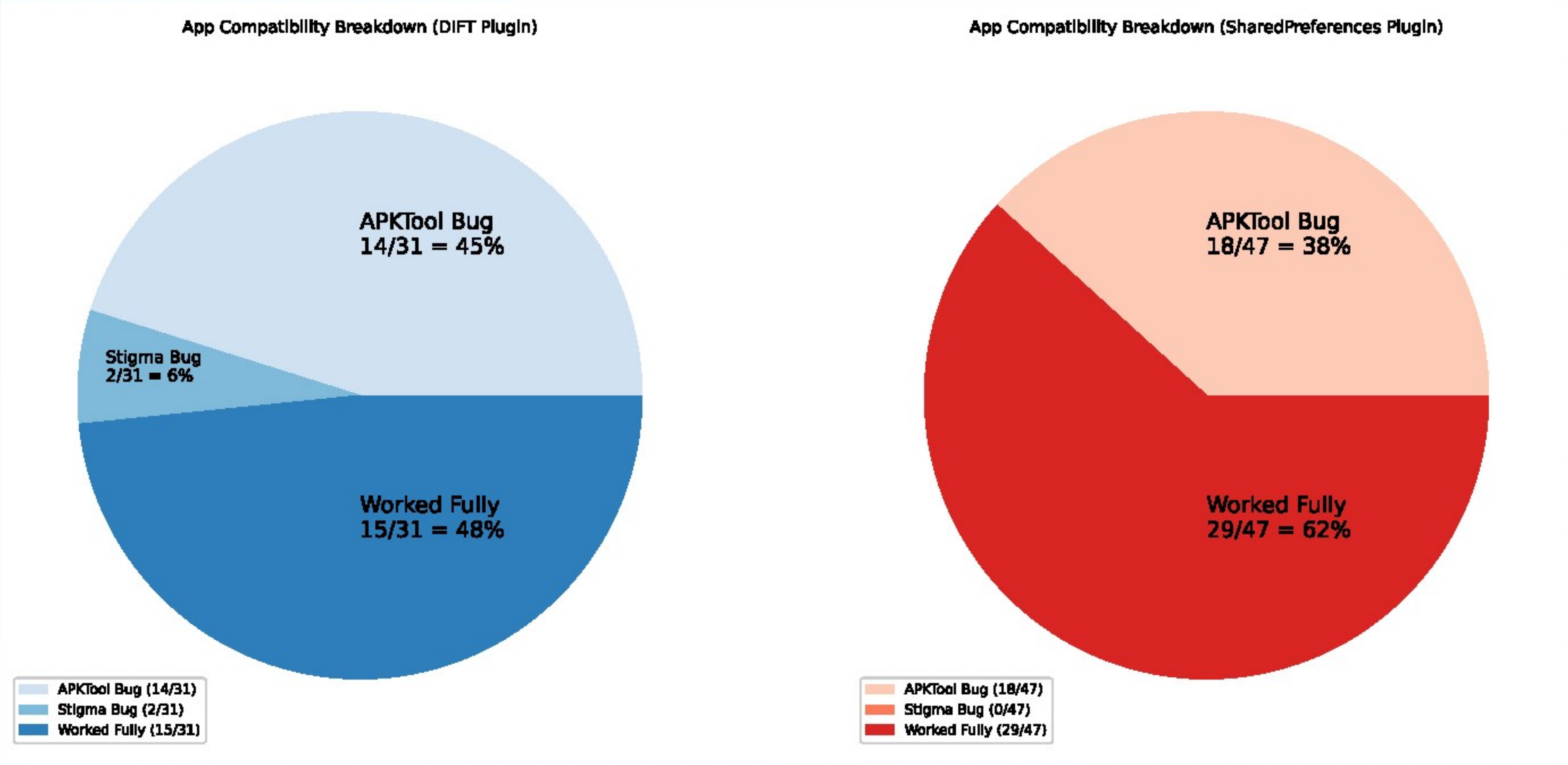
- Smali assembly code acquisition
  - Apktool, unattainable code

- Smali code has numerous concepts and technical rules that are minimally documented
  - Virtual methods and Java “syntax sugar” (e.g. lambda functions)
  - Register allocation
  - Type system and type-specific instructions (`move` vs. `move-wide`)
  - Multi-line instructions
  - Unintentional control-flow modifications (potential exceptions in try/catch blocks)
  - Code offsets
  - Reference pool limits (classes.dex, classes2.dex, classes3.dex etc.)

- App repackaging & cryptographic signatures

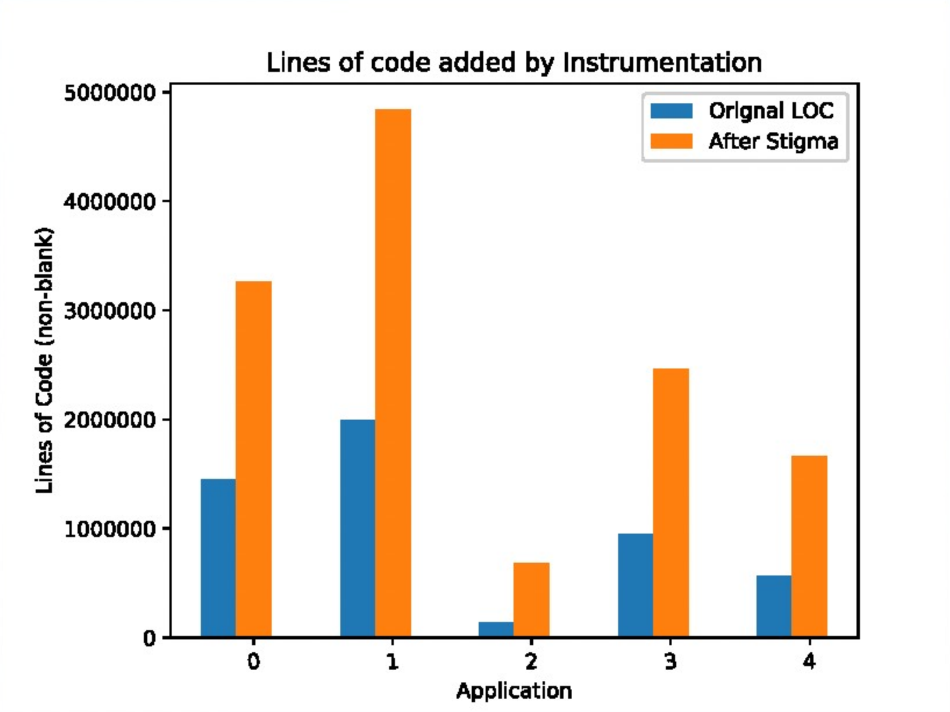
Evaluation

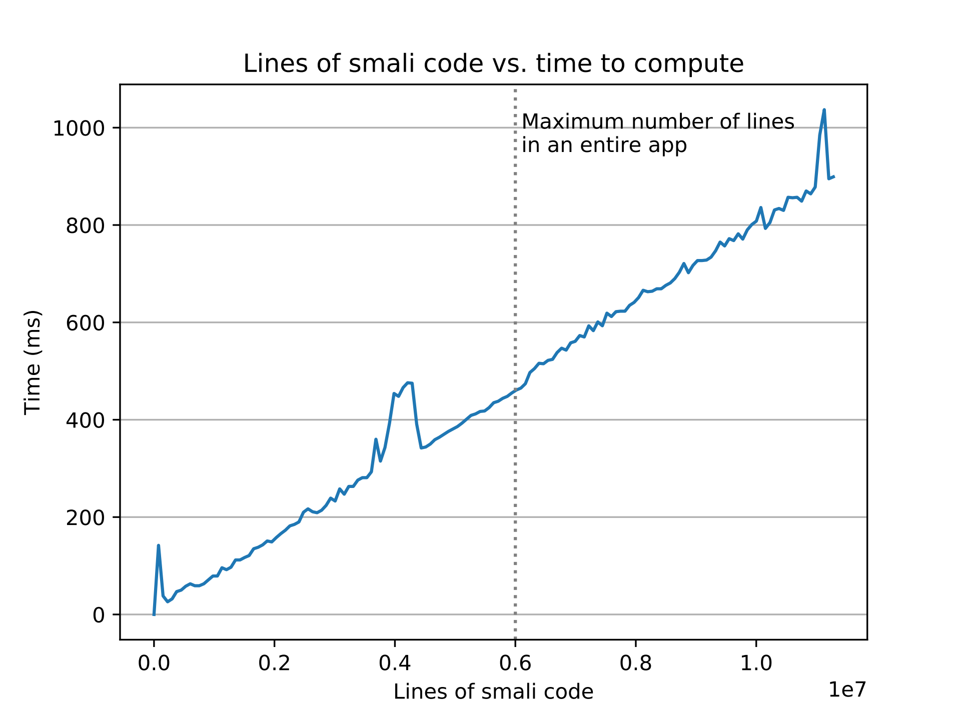
Broad Compatibility



Evaluation

Overhead





References and Similar Projects

- Stigma Project Source Code: https://github.com/fmresearchnovak/stigma
- Associated Technical Report: https://ednovak.net/documents/stigma_tr.pdf
- Apktool: https://apktool.org/
- Smali Project: https://github.com/JesusFreke/smali

- Some Similar Projects and Papers
  - Cybia Substrate: https://www.cydiasubstrate.com/
  - DDI: https://github.com/crmulliner/ddi
  - ViaLin: https://resess.github.io/artifacts/ViaLin/
  - SiF: https://dl.acm.org/doi/10.1145/2462456.2465430
  - Dr. Android and Mr. Hide: https://dl.acm.org/doi/10.1145/2381934.2381938

Future Work

- Improving Compatibility w/Apps
  - Upstream contributions to apktool
- Writing New Plugins
  - Removing ADs
  - Identifying Security Vulnerabilities
  - Automatic bug fixing
- Using A.I. to generate smali code / smali modifications
- Improving the UI/UX and general usability