Static Asynchronous Component Misuse Detection for Android Applications

ESEC/FSE 2020

Linjie Pan, Baoquan Cui, Hao Liu, Jiwei Yan, Siqi Wang,
Jun Yan and Jian Zhang

Institute of Software, Chinese Academy of Sciences
Presenter: **Linjie Pan**

Contents

- Introduction
- Misuse Pattern
- Approach
- Evaluation
- Conclusion

Android Asynchronous Programming

- Android adopts single-thread model
  - Main thread: GUI update
  - Background thread: time-consuming or CPU (IO)-blocking task
- Android provides packaged components to simplify async programming
  - AsyncTask
  - IntentService
  - HandlerThread
  - …
- How do developers use these components?

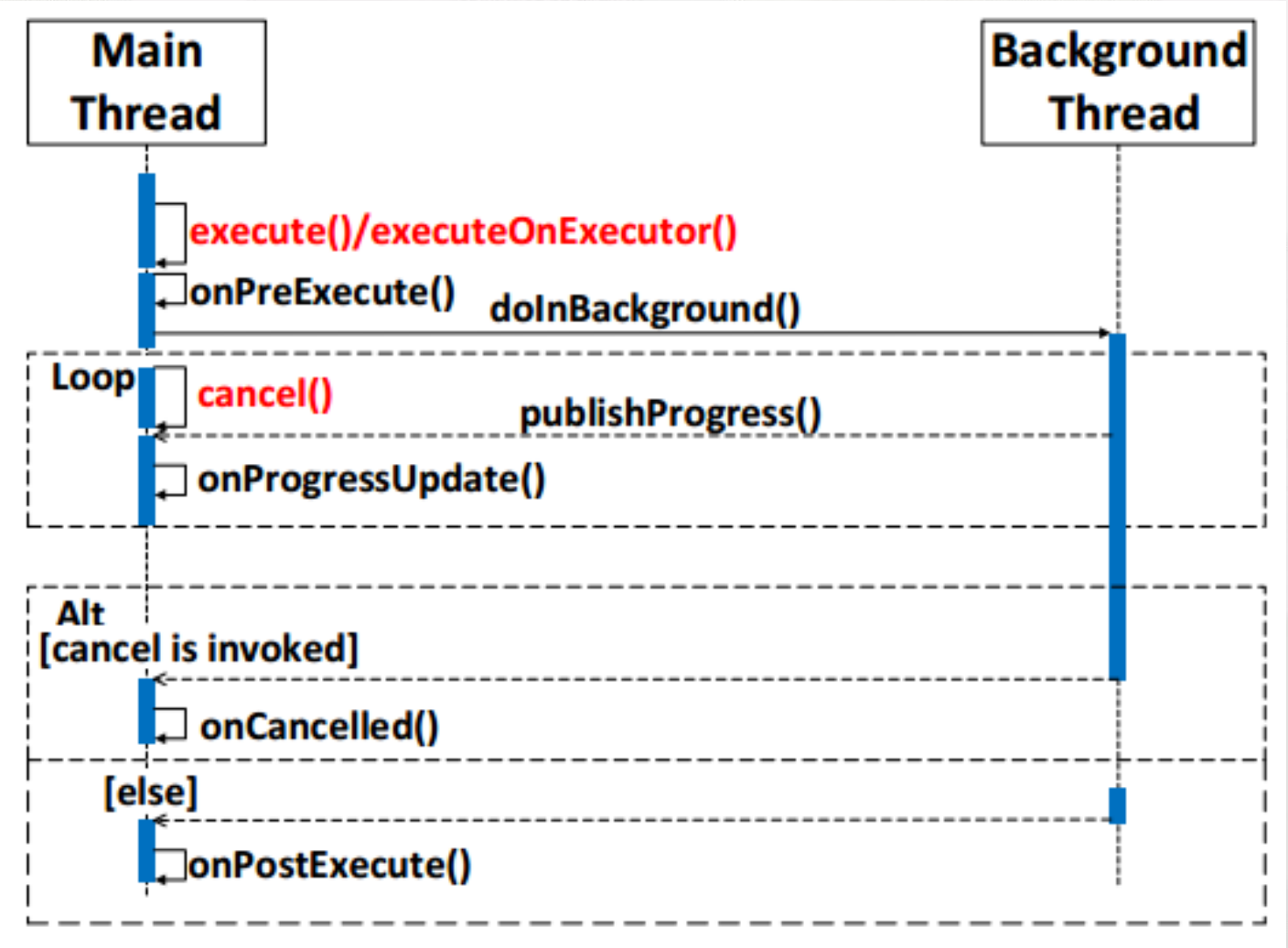
Motivation

- Empirical study on three repositories: F-Droid(1184), GooglePlay(6808), Wandoujia(5669)
- The improper use of AsyncTask can lead to many problems
  - Memory leak
  - Wrong update
  - …
- We have developed a static analysis tool called AsyncChecker to detect misuse of AsyncTask

Async Components	F-Droid			GooglePlay			Wandoujia		
	Class	App	Perc(%)	Class	App	Perc(%)	Class	App	Perc(%)
AsyncTask	2,482	421	35.6	117,295	5,660	83.1	88,010	3,779	66.7
IntentService	190	118	10.0	12,501	3,535	51.9	8,902	2,299	40.6
HandlerThread	63	60	5.1	4,096	2,420	35.5	2,906	1,487	26.2
AsyncTaskLoader	83	23	1.9	797	173	2.5	586	127	2.2
ThreadPoolExecutor	125	111	9.3	8,526	3,714	54.6	3,548	1,966	34.7
Thread	1,299	367	31.0	58,426	5,524	81.1	118,084	3,413	60.2

Background of AsyncTask

- onPreExecute
- doInBackground
- onProgressUpdate
- onCancelled
- onPostExecute



Contents

- Introduction
- Misuse Pattern
- Approach
- Evaluation
- Conclusion

Misuse Pattern

- StrongReference
- NotCancel
- NotTerminate
- EarlyCancel
- RepeatStart

Example of StrongReference

- An instance of AsyncTask holds strong reference to Activity
- AsyncTask lives longer than Activity
- The memory of Activity cannot be garbage collected

```
1  public class MainActivity extends Activity {
2      private TextView view1;
3      private AsyncTask task;
4      protected void onCreate(Bundle bundle) {
5          super.onCreate(bundle);
6          view1 = (TextView) findViewById(R.id.view1);
7          task = new Tasker(view1); // holding strong reference
8          task.execute();
9      }
10     protected void onDestroy() {
11         super.onDestroy();
12         task.cancel();
13     }
14 }
15 class Tasker extends AsyncTask {
16     private TextView view2;
17     public Tasker(TextView view1) {
18         view2 = view1;
19     }
20      ...
21 }
```

Example of NotCancel

- AsyncTask is not canceled before the destruction of Activity
- *onPostExecute()* contains GUI update operation
- Wrong update

```
1  public class MainActivity extends Activity {
2     private ProgressDialog searchDialog;
3     private AsyncTask task;
4     protected void onCreate(Bundle bundle) {
5        ...
6        task = new Tasker(searchDialog);
7        task.execute();
8     }
9     protected void onDestroy() {
10       super.onDestroy();
11       // task should be canceled
12    }
13    private static class Tasker extends AsyncTask {
14       private WeakReference<ProgressDialog> dialog;
15       public Tasker(ProgressDialog theDialog) {
16          dialog = theDialog;
17       }
18       protected void onPostExecute() {
19          dialog.dismiss(); // potential crash
20       }
21       protected String doInBackground(String[] arg) {
22          ...
23       }
24       ...
25    }
26 }
```

Example of NotTerminate

- AsyncTask can’t be terminated after *cancel()* is invoked
- Check *isCancelled()* periodically within a loop in *doInBackground()*

```
1  class Tasker extends AsyncTask {
2     protected String doInBackground(String[] arg) {
3        while ( ... ) {
4           // loop should terminate if cancel() is invoked
5        }
6        return "message";
7     }
8  }
```

Example of EarlyCancel and RepeatStart

- EarlyCancel

```
protected void onCreate(Bundle bundle) {
   ...
   task = new Tasker();
   task.cancel(true); // task is cancelled before it is
        started
   task.execute();
   ...
}
```

- RepeatStart

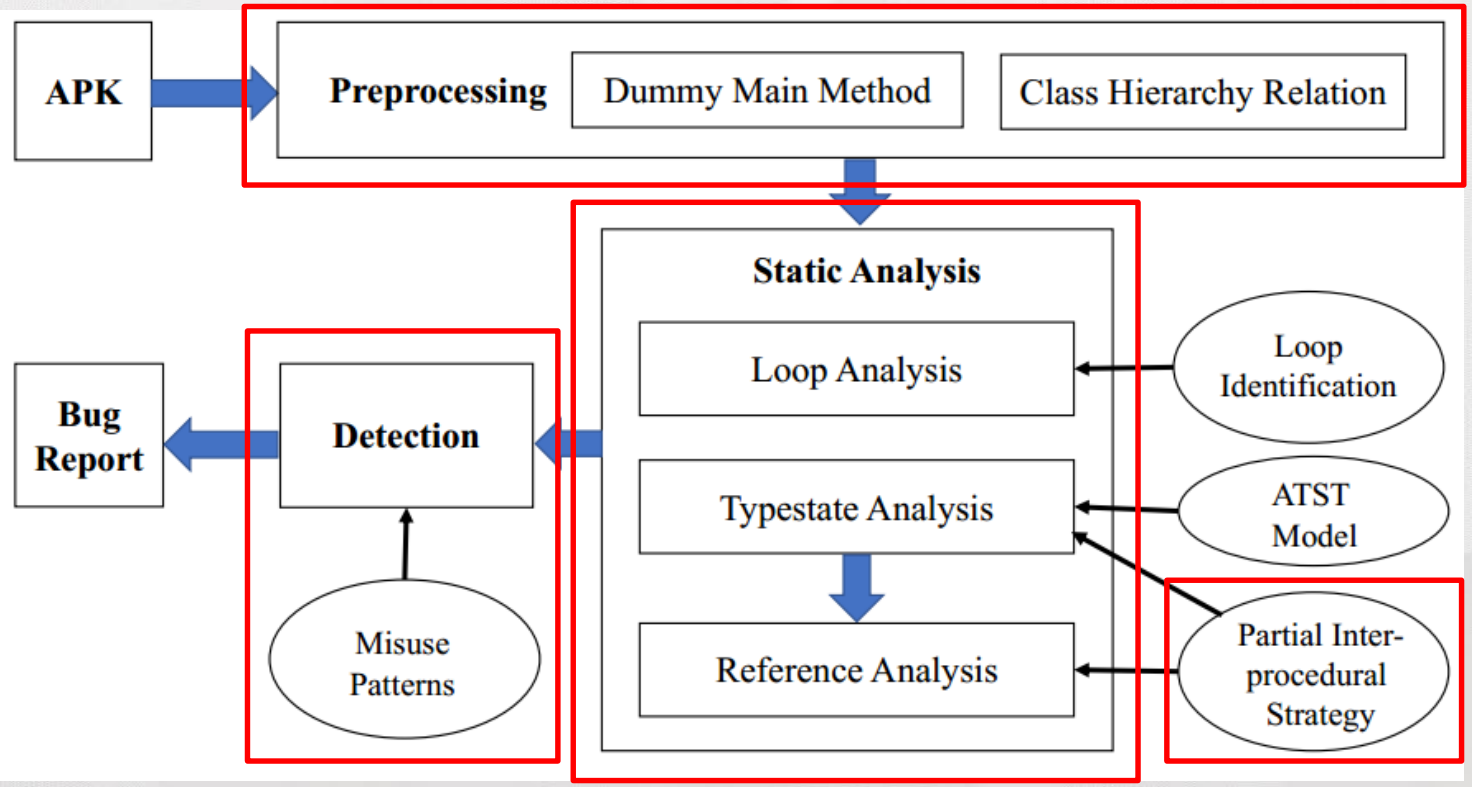
```
class MainActivity extends Activity {
   private Tasker task;
   protected void onCreate(Bundle bundle) {
      task = new Tasker();
      ...
   }
   protected void onStart() {
      task.execute(); // task will execute twice if user
           navigates back
      ...
   }
}
```

Contents

- Introduction
- Misuse Pattern
- Approach
- Evaluation
- Conclusion

Overview of AsyncChecker

- Typestate Analysis (NotCancel, EarlyCancel, RepeatStart)
- Reference Analysis (StrongReference)
- Loop Analysis (NotTerminate)

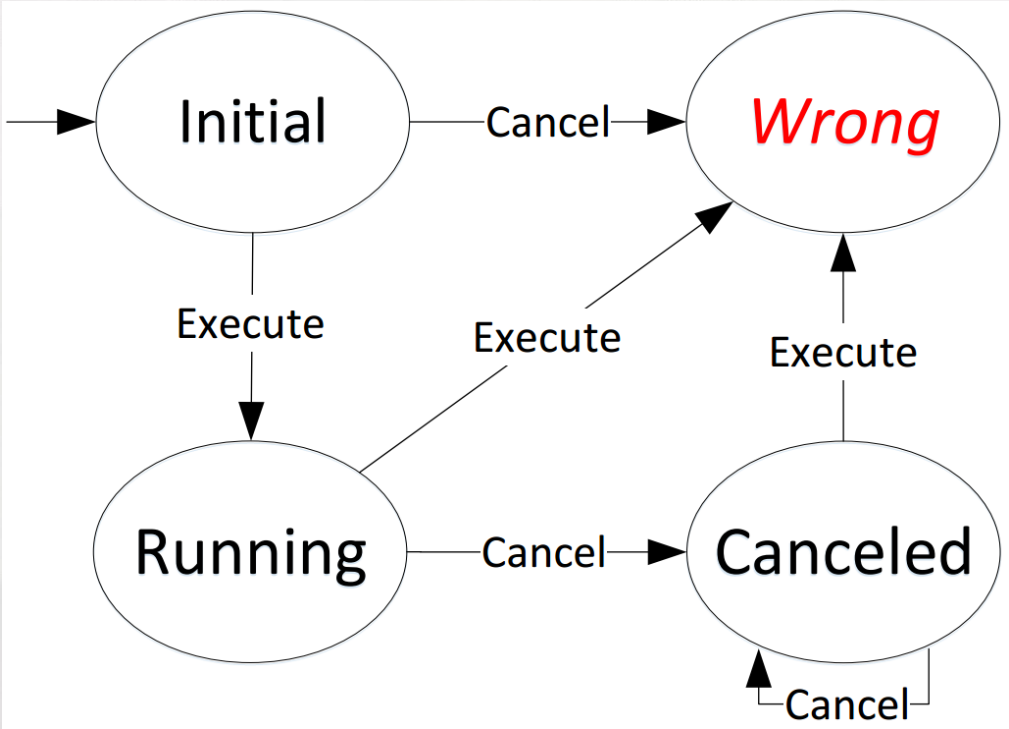


Partial Inter-procedural Strategy

- Most of the operations in an app are AsyncTask-irrelevant
- Only perform inter-procedural analysis for methods that contain AsyncTask-related statements
- May interprocedural-analysis for call site
- Traverse all possible invoked methods

Typestate Analysis (NotCancel, EarlyCancel, RepeatStart)

- A typestate denotes the state an object can occupy during execution
- The operations on an AsyncTask object such as *execute()* and *cancel()* can change its state
- Build the AsyncTask State Transition (ATST) model
- TypeState analysis is also the basis of reference analysis



Reference Analysis (StrongReference)

- Build the mapping relation from the fields of AsyncTask to the GUI objects
- Possession operation (AssignStmt)
  - Left: field of AsyncTask
  - Right: GUI object
- The type of an GUI object is the subclass of View or Activity
- WeakReference or reference to non-GUI object does not lead to StrongReference

```
1  class MainActivity extends Activity {
2     protected void onCreate(Bundle bundle) {
3        TextView view1 = (TextView) findViewById(R.id.view1);
4        TextView view2 = (TextView) findViewById(R.id.view2);
5        String mark = "tasker";
6        Tasker task = new Tasker(view1, mark);
7        task.setView2(view2);
8        task.execute("begin"); // AsyncTask is started
9     }
10 }
11 class Tasker extends AsyncTask<String,String,String> {
12    private TextView v1;
13    private WeakReference<TextView> v2;
14    private String s1;
15    public Tasker(TextView tv1, String mark) {
16       this.v1 = tv1; // strong reference to GUI object
17       this.s1 = mark; // strong reference to non-GUI object
18    }
19    public void setView2(TextView tv2) {
20       this.v2 = new WeakReference<TextView>(tv2); // weak
               reference to GUI object
21    }
22 }
```

Loop Analysis (NotTerminate)

- Identify loop structure and the invocation of *isCancelled()* in *doInBackground()*
- Perform loop analysis in bottom-up manner via topological sort
- Use a set to save loops that do not contain terminate operation

```
1  class Task extends AsyncTask<String,String,String> {
2     protected String doInBackground(String[] args) {
3        for (int i = 0; i < 100; i++) { // First loop
4           if (isCancelled()) break; // Terminate operation
5           ...
6        }
7        theLoop();
8     }
9     private void theLoop() {
10       for (int j = 0; j < 100; j++) { // Second loop
11          ... // Lack Terminate operation
12       }
13    }
14    ...
15 }
```

Implementation

- AsyncChecker is based on Androlic [1]
- Analyze the semantics of statements via the core engine of Androlic
- Implement some APIs of Androlic
  - IMethodInterProceduralJudge (Partial inter-procedural strategy)
  - NewRefHeapObject (AsyncTaskHeapObject)
  - AbstractTypeState (AsyncTypeState)
  - ILibraryInvocationProcessor (identify AsyncTask operation)
- AsyncChecker is open source (**https://github.com/pangeneral/AsyncChecker**)

[1] Linjie Pan, et al. Androlic: an extensible flow, context, object, field, and path-sensitive static analysis framework for Android. In ISSTA 2019.

Contents

- Introduction
- Misuse Pattern
- Approach
- Evaluation
- Conclusion

Evaluation

- **RQ1:** How effective is AsyncChecker on self-designed benchmark?
- **RQ2:** Can AsyncChecker find the misuse of AsyncTask in real-world applications?
- **RQ3:** How effective is AsyncChecker against existing tools?
- **RQ4:** Do developers take the misuse of AsyncTask as a serious problem?

AsyncChecker on Benchmark

- Design an AsyncTask-specifc benchmark suite called AsyncBench
- AsyncBench contains 69 manually written Android apps

Benchmark Group	TP	TN	FP	FN
StrongReference	2	3	0	2
NotCancel	8	4	0	0
NotTerminate	8	7	0	0
EarlyCancel	6	6	0	1
RepeatStart	8	0	0	2
Combination	13	3	0	4
Sum	45	23	0	9

AsyncChecker on Real-world Applications

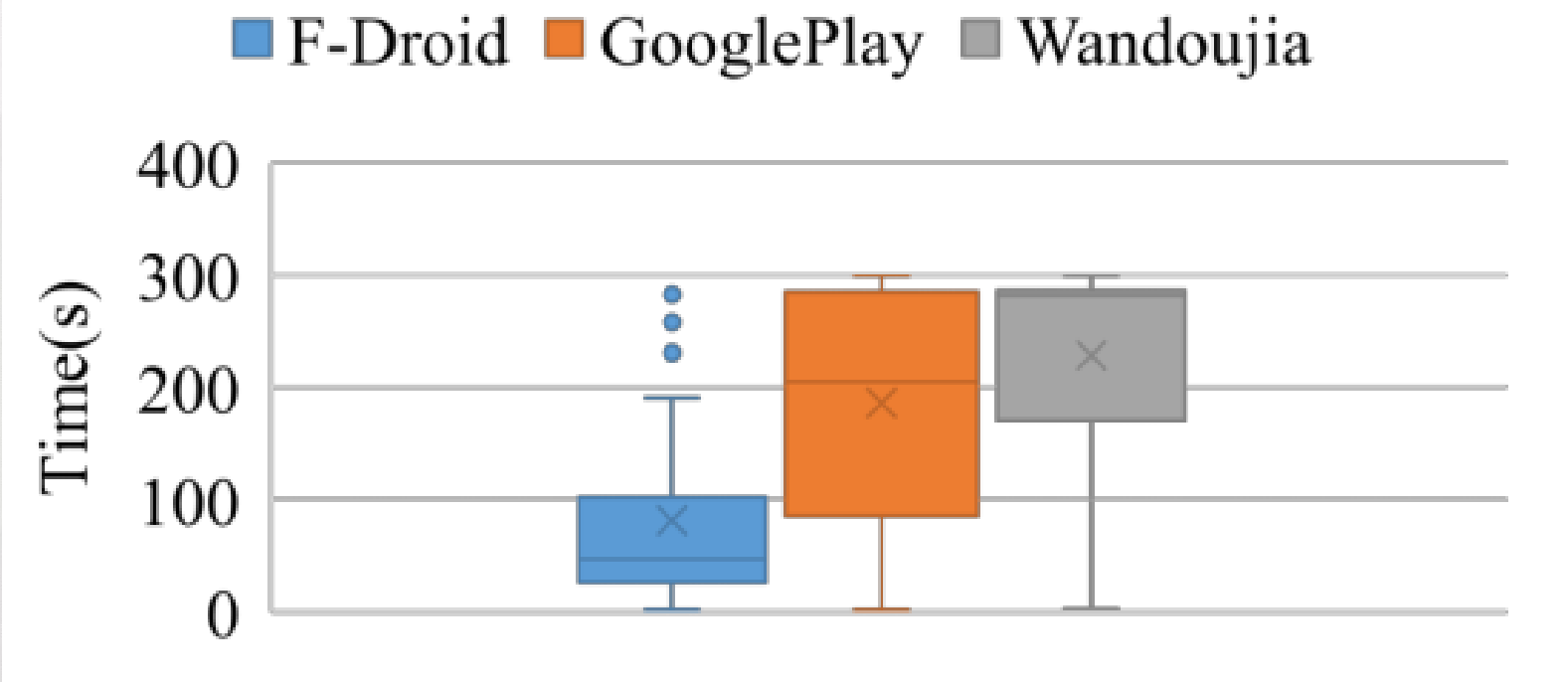
- Among 1759 real-world apps, 1417 (80.6%) contain at least one misused AsyncTask instance
- The frequency of EarlyCancel and RepeatStart are lower than rest patterns

Respository	Inst		App	
	Correct	Misused	Correct	Misused
F-Droid	50	416	13	58
GooglePlay	4,033	13,098	178	931
Wandoujia	2,107	4,432	151	428
Sum	6,190	17,946	342	1,417

Misuse Pattern	F-Droid		GooglePlay		Wandoujia		Sum	
	Inst	App	Inst	App	Inst	App	Inst	App
StrongReference	261	50	4,901	742	1,613	300	6,775	1,092
NotCancel	354	55	9,108	842	3,540	394	13,302	1,291
NotTerminate	304	58	10,916	881	3,496	391	14,716	1,330
EarlyCancel	2	2	23	16	11	8	36	26
RepeatStart	3	3	24	6	2	2	36	11

Efficiency of AsyncChecker

- Average running time is 82 (F-Droid), 198 (GooglePlay) and 228 (Wandoujia) seconds respectively



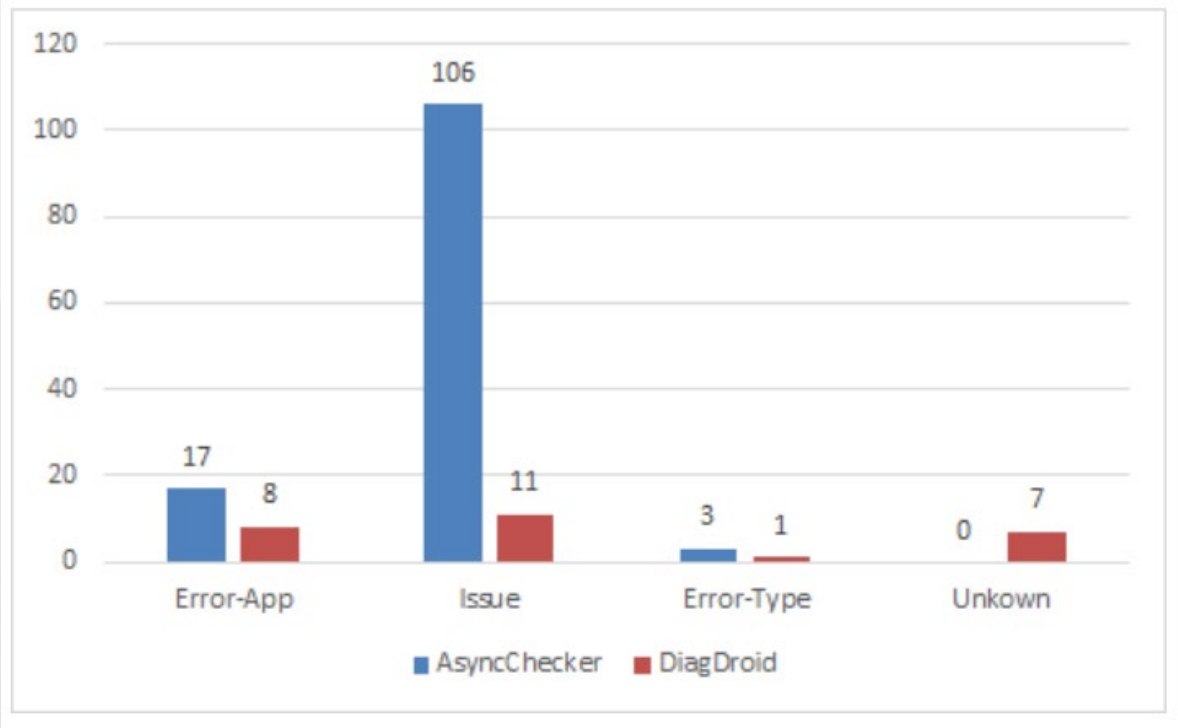
Validation

- Randomly select 22 apps from F-Droid
- Precision, Recall and F-measure are 97.2%, 89.8% and 0.93 respectively

App Name	StrongReference			NotCancel			NotTerminate		
	TP	FP	FN	TP	FP	FN	TP	FP	FN
AFWall [1]	2	0	0	2	0	0	2	0	0
AnkiDroid [2]	0	0	0	3	0	0	3	0	0
A Photo Manager [3]	2	0	0	0	0	2	2	0	0
Easy xkcd [13]	0	0	0	1	0	1	1	0	0
Kiss [28]	0	0	0	1	0	1	2	0	0
Minetest [38]	1	0	0	0	0	0	1	0	0
Mythmote [39]	1	0	0	1	1	0	1	0	0
NextCloud [40]	2	0	0	2	0	0	2	0	0
OpenBikeSharing [41]	1	0	0	1	0	0	1	0	0
Password Store [55]	2	0	0	2	0	0	2	0	0
rootless-logcat [49]	1	0	0	0	0	0	0	0	0
SatStat [50]	0	0	0	0	0	0	1	0	0
Seafile [51]	12	0	2	10	2	5	13	0	0
Simple Gallery [53]	0	0	0	0	0	1	1	0	0
syncthing-android [57]	1	0	0	2	0	0	2	0	0
Web Opac App [4]	3	0	0	2	0	0	2	0	0
Wikimedia Commons [9]	3	0	0	7	0	0	5	0	0
Sum	31	0	2	34	3	10	41	0	0

Comparison with Existing Tools

- Existing tools: APEChecker [1] and DiagDroid [2]
- Obtain source code of DiagDroid
- Compare the results of DiagDroid and AsyncChecker on 22 apps from F-Droid



[1] Lingling Fan, Ting Su, Sen Chen, Guozhu Meng, Yang Liu, Lihua Xu, and Geguang Pu. Efficiently manifesting asynchronous programming errors in Android apps. In ASE'18, Montpellier, France, September 3-7. 486–497.
[2] Yu Kang, Yangfan Zhou, Hui Xu, and Michael R. Lyu. 2016. DiagDroid: Android Performance Diagnosis via Anatomizing Asynchronous Executions. In FSE'16, Seattle, WA, USA, November 13-18. 410–421.

Feedback from Developers

- Report issues of 17 apps to developers of F-Droid applications via GitHub
- Most of the problems that have been fixed are caused by StrongReference

App Name	Stars	Issue ID	Commit ID	Result
AnkiDroid	2,023	5404	/	confirmed
A Photo Manager	111	145	7fb4c03	fixed
Easy xkcd	101	164	/	confirmed
Kiss	1,281	/	fa5ab40	fixed
Minetest	4,365	8787	/	confirmed
Password Store	1,271	/	56e53d3	fixed
rootless-logcat	59	15	/	confirmed
syncthing-android	1,157	1142	b93da52	fixed
Web Opac App	108	566	/	confirmed
Wikimedia Commons	519	2791	/	confirmed

Contents

- Introduction
- Misuse Pattern
- Approach
- Evaluation
- Conclusion

Conclusion

- Summarize five misuse patterns of AsyncTask
- Develop a static analysis tool called AsyncChecker
- Misuses of AsyncTask exists in real-world apps (up to 80.6% in 1,759 apps)

Thank you!