ServDroid: Detecting Service Usage Inefficiencies in Android Applications

Summary

- Introduction
- Background
- Service Usage Anti-Patterns
- Detecting Service Usage Bugs
- Empirical Evaluation
- Conclusion

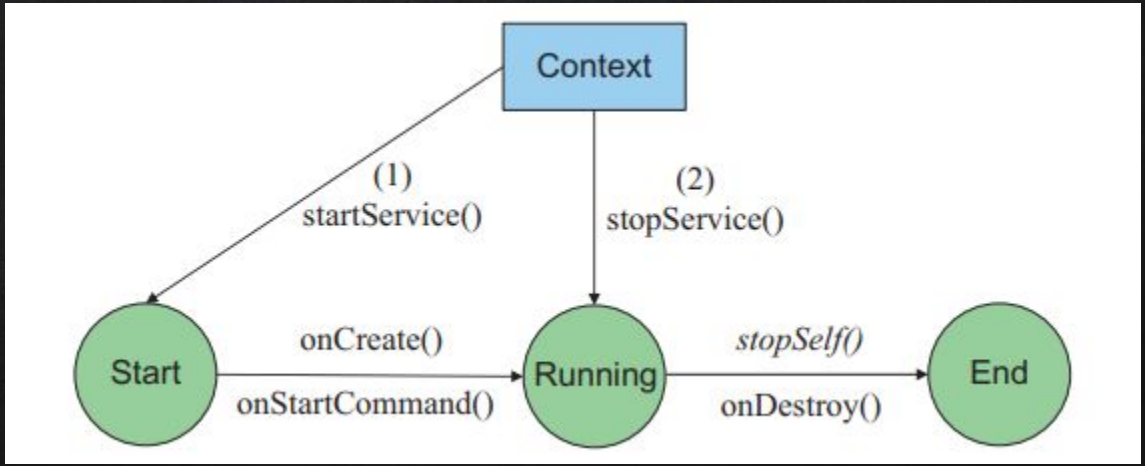
Introduction

Introduction

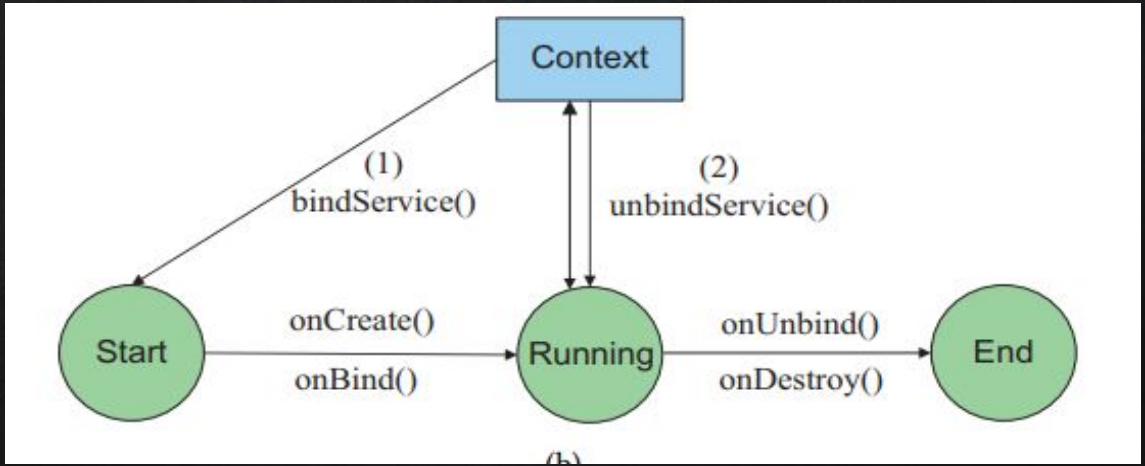
- System Services & App Services
- Started Services, Bound Services, Hybrid Services
- Services Anti-Patterns
- Unneeded Memory Allocation and Energy Inefficiencies
- ServDroid

Background

Started Service



Bound Service



Hybrid Service

Started Service

+

Bound Service

Services Usage Anti-Patterns

Services Usage Anti-Patterns

- Premature create
- Late Destroy
- Premature Destroy
- Service Leak

Anti-pattern \ Service type	Started	Bound	Hybrid
Premature create		√	√
Late destroy	√	√	√
Premature destroy	√		√
Service leak	√	√	√

Premature Create

```
1  public class NCBlackListActivity{
2    public final void run(){
3      Intent intent = new Intent(context, NotificationManagerService
4      .class);
5      context.bindService(intent, aqL.dYV, 1);
6      oj();
7      arF();
8       ...
9      dZm.aqC();
10   }
11 }
12 public class NotificationManagerService extends Service {
13   public IBinder onBind(Intent intent) {
14       ...
15     return this.dZm;
16   }
17 }
```

Premature Create

```
1  public final class GoogleDriveActivity extends apf implements
2    c.b, e.a{
3      public final void onCreate(Bundle paramBundle){
4        Intent localIntent = getIntent();
5        onNewIntent(localIntent);
6        q();
7        . . . . . .
8        getApplicationContext().bindService(new Intent(this,
9        GoogleDriveService.class), this.af, 1);
10     }
11     protected final void onNewIntent(Intent paramIntent){
12       m();
13     }
14     final void m(){
15       Intent localIntent = new Intent(this, GoogleDriveService.class);
16       getApplicationContext().startService(localIntent);
17     }
18   }
```

Services Usage Anti-Patterns

- Premature create
- Late Destroy
- Premature Destroy
- Service Leak

Anti-pattern \ Service type	Started	Bound	Hybrid
Premature create		√	√
Late destroy	√	√	√
Premature destroy	√		√
Service leak	√	√	√

Late Destroy

```
1  public class OverlayService extends Service{
2    public static void a(Context paramContext){
3      paramContext. startService (new Intent(paramContext, OverlayService.
4          class ));
5    }
6    public int onStartCommand(Intent paramIntent, int paramInt1, int
7      paramInt2){
8      return;
9    }
10   public static void b(Context paramContext){
11     paramContext.stopService(new Intent(paramContext, OverlayService.
12         class ));
13   }
14 }
```

Late Destroy

```
1  final class ahx extends agj implements ServiceConnection{
2    final void b(){
3      d();
4      . . . . . .
5      f();
6      c();
7      . . . . . .
8      e();
9    }
10   final void d(){
11     localIntent = new Intent("android.media.
12     MediaRouteProviderService");
13     this.o = this.a.bindService( localIntent , this , 1);
14   }
15   final void f(){
16     localahy.h.j.post(new ahz(localahy));
17   }
18   final void e(){
19     this.a.unbindService( this );
20   }
21 }
```

Services Usage Anti-Patterns

- Premature create
- Late Destroy
- Premature Destroy
- Service Leak

Anti-pattern \ Service type	Started	Bound	Hybrid
Premature create		√	√
Late destroy	√	√	√
Premature destroy	√		√
Service leak	√	√	√

Premature Destroy

```
1  public class MessageService extends Service{
2    public void a(Context paramContext){
3      Intent intent=new Intent(paramContext.this, MessageService.class);
4       startService (intent);
5    }
6    public void b(Context paramContext){
7      Intent intent=new Intent(paramContext.this, MessageService.class);
8       startService (intent);
9    }
10   public void m(Context paramContext){
11     Intent intent=new Intent(paramContext.this, MessageService.class);
12      startService (intent);
13   }
14   public int onStartCommand(Intent paramIntent,int paramInt1,
15     int paramInt2){
16        . . . . . .
17        stopSelf ();
18        return 1;
19     }
20 }
```

Services Usage Anti-Patterns

- Premature create
- Late Destroy
- Premature Destroy
- Service Leak

Anti-pattern \ Service type	Started	Bound	Hybrid
Premature create		√	√
Late destroy	√	√	√
Premature destroy	√		√
Service leak	√	√	√

Service Leak

```
1  public class MpActivity extends Activity {
2    public final void a(g paramg){
3      startService (new Intent( this , MpService.class ));
4       . . . . .
5      if (b() ){
6        stopService (new Intent( this , MpService.class ));
7      }
8      else {
9          . . . . .
10     }
11   }
12 }
```

Service Leak

```
1  public class ArtMonitorImpl{
2      public void startMonitoring (){
3          ...
4          this .mContext.bindService(new Intent( this .mContext,
5          ArtDownloadService.class ),  this .mServiceConnection, 5);
6          ...
7      }
8  }
```

(b)

Detecting Service Usage Bugs

Detecting Service Usage Bugs

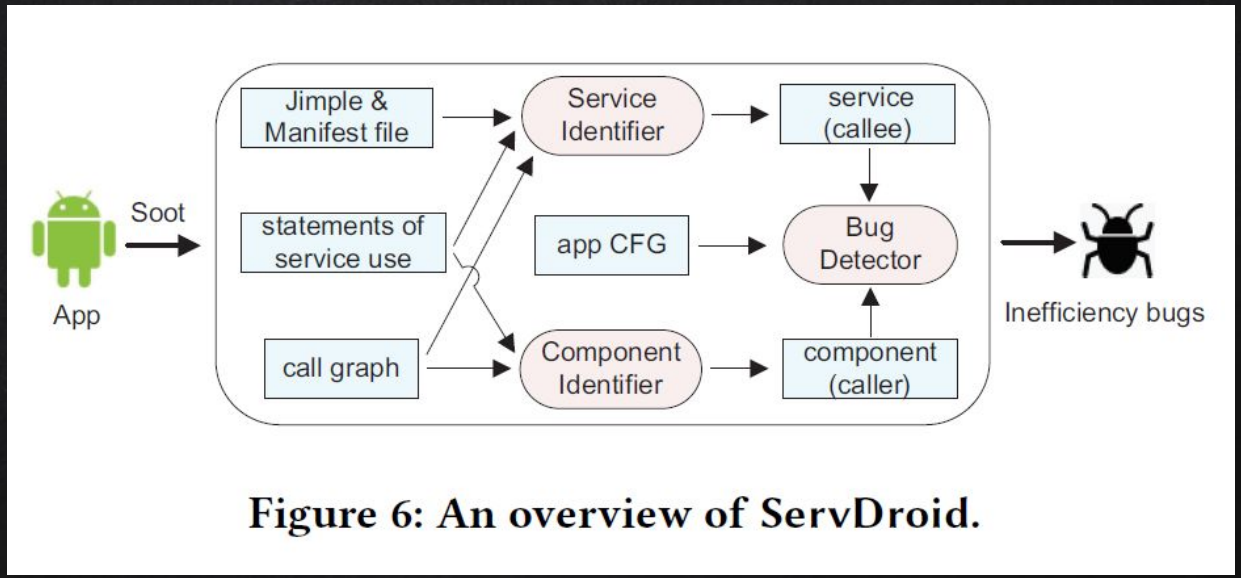


Figure 6: An overview of ServDroid.

Call Graph generated by Soot Framework

Detecting Service Usage Bugs

- Info Flow CFG

Definition 1 (Dominator). *In a CFG, a node (statement) s_j is dominated by another node s_i if every path from the entry of the CFG to s_j contains s_i. s_i is called a dominator of s_j.*

Definition 2 (Post-dominator). *In a CFG, a node (statement) s_i is post-dominated by another node s_j if every path from s_i to the exit of the CFG contains s_j. s_j is called a post-dominator of s_i.*

App Control Flow CFG is also generated by Soot

Detecting Premature Create Bugs 1

1. Search in every method of app all statements stmB which is a *bindService()*;
2. For every stmB determine c (client component) and s (service);
3. Then, from CFG we find a post-dominator stmU which is the first statement that invokes a method of s;
4. We see if there is any other statement between stmB and stmU. If there is, there is a premature create;

Detecting Premature Create Bugs 2

1. Search in every method of app all statements stmB which is a *bindService()*;
2. For every stmB determine c (client component) and s (service);
3. Check if service s doesn't overwrite *onStartCommand()*;
4. If it doesn't, the CFG must satisfy these two conditions:
    a. No component binds service when *startService()* executes;
    b. The *bindService* statement is not a immediate post-dominator of *startService()* there is no *stopService()* between them;

Detecting Late Destroy Bugs

- For started services:
  - Detection is straightforward, and it's reduced to check if *stopService()* or *stopService(int)* is not called in the *onStartCommand()*;

Detecting Late Destroy Bugs

- For bound services:
  1. For each *unbindService()* we determine c (client component) and s (service) such that c unbinds s;
  2. In CFG, we look for stmU such that stmU is the last call of a method of s from c;
  3. If there are statements between then, there is a lat destroy;

Detecting Premature Destroy Bugs

If the started or hybrid service is shared by two or more components (callers) and *stopSelf()* is called instead of *stopSelf(int)* in the *onStartCommand()* we have a premature destroy;

Detecting Service Leak Bugs

1. Find all start (bind) statements and put them in set S1;
2. Find all stop (unbind) statements and put them in set S2;
3. Remove from S2 all statements that are triggered by end users;
4. For each start (bind) in S1, check whether its corresponding stop (unbind) statement is on S2. If it's not then service leaks, if yes, we check we check if the stop statement is always reached by the start statement. If not, service leaks;

Empirical Evaluation

1.

What are the precision, recall, and time overhead of ServDroid?

- Empirical Study on 45 Apps
- Manual vs ServDroid

App name	Version	# Services				# Service usage inefficiency bugs				
		Started	Bound	Hybrid	Total	PCBs	LDBs	PDBs	SLBs	Total
Google Play services	11.0.55 (436-156917137)	**130**	**17**	6	**153**	**6**	4	0	**95**	**105**
Gmail	7.6.4.158567011.release	18	2	4	24	1	1	0	5	7
Maps	9.54.1	12	9	4	25	2	1	1	9	13
YouTube	12.23.60	10	2	3	15	2	2	0	3	7
Facebook	10.2.0	22	8	4	34	0	4	0	5	9
Google	7.3.25.21.arm	21	10	6	37	5	3	0	1	9
Google+	9.14.0.158314320	23	1	4	28	0	1	0	7	8
GoogleText-to-Speech	3.11.12	4	1	0	5	0	0	0	2	2
WhatsApp Messenger	2.17.231	9	5	5	19	5	2	**2**	10	19
Google Play Books	3.13.17	8	2	1	11	0	0	0	1	1
Messenger	123.0.0.11.70	40	1	0	41	0	1	0	9	10
Hangouts	20.0.156935076	11	9	2	22	0	1	1	4	6
Google Chrome	58.0.3029.83	16	9	2	27	0	0	0	0	0
Google Play Games	3.9.08(3448271-036)	4	0	0	4	0	0	0	3	3
Google TalkBack	5.2.0	2	1	0	3	0	0	0	1	1
Google Play Music	7.8.4818-1.R.4063206	15	3	4	22	1	3	1	10	15
Google Play Newsstand	4.5.0	8	1	1	10	0	1	1	3	5
Google Play Movies & TV	3.26.5	6	4	2	12	0	0	0	4	4
Google Drive	2.7.153.14.34	6	6	3	15	1	2	0	3	6
Samsung Push Service	1.8.02	11	0	0	11	0	3	0	1	4
Instagram	10.26.0	14	4	2	20	3	1	0	7	11
Sum	-	601	186	105	892	55	90	19	318	482
Average	-	13.4	4.1	2.3	19.8	1.2	2.0	0.4	7.1	10.7

1.

What are the precision, recall, and time overhead of ServDroid?

- Empirical Study on 45 Apps
- Manual vs ServDroid

Recall and Precision: 100%

Time Overhead: 4,3 s

Time to analyze per app: 96 s

2.

How much energy can be saved if these services usage inefficiency bugs are fixed?

- Fix the bugs
- Original vs Repackaged apps for 15 minutes each
- Trepn Profiler

App name	Energy consumption	
	Original (J)	Repaired (J)
Google Play services	574.68	373.42
Gmail	437.05	388.25
Maps	626.67	487.28
YouTube	803.31	687.05
Facebook	553.94	510.34
Google	498.40	426.82
Google+	438.59	422.57
GoogleText-to-Speech	395.90	353.91
WhatsApp Messenger	421.93	343.58
Google Play Books	474.77	435.57
Messenger	509.60	412.92
Hangouts	370.33	338.30
Google Play Games	549.86	475.03
Google TalkBack	458.57	429.97
Google Play Music	567.35	458.59
Google Play Newsstand	512.50	442.20
Google Play Movies & TV	363.39	317.45
Google Drive	338.49	286.74
Instagram	496.44	452.05

2.

How much energy can be saved if these services usage inefficiency bugs are fixed?

- Fix the bugs
- Original vs Repackaged apps for 15 minutes each
- Trepn Profiler

Average Consumption: 546, 7 J –> 459,5 J = 87,14 J (– 15.94%)

3.

Are background services widely used in Android apps? Which type of services are used most frequently?

- 1,000 apps:
  - 939 use background services.
  - Started Service: 4,952 (60.87%)
  - Bound Service: 2,468 (30.34%)
  - Hybrid Service: 715 (8.79%)

8 services on average per App

4.

Are service usage inefficiency bugs common in practice?

- 825 (82.5%) have at least one kind of inefficiency bug;
- 608 (60.8%) have at least two kinds of inefficiency bugs;
- 304 (30.4%) have no less than three kinds of inefficiency bugs;
- and 59 (5.9%) have all the four kinds of inefficiency bugs.

Each app has 4.43 service usage inefficiency bugs on average

5.

How are the four kinds of service usage inefficiency bugs distributed in the three types of services?

	Started	Bound	Hybrid
Premature Create	0 (0%)	400 (71,05%)	163 (28.95%)
Late Destroy	491 (38.18%)	620 (48.21%)	620 (48.21%)
Premature Destroy	137 (78.29%)	0 (0%)	38 (21.71%)
Service Leak	1,720 (71.61%)	392 (16.32%)	290 (12.07%)

Anti-pattern \ Service type	Started	Bound	Hybrid
Premature create		√	√
Late destroy	√	√	√
Premature destroy	√		√
Service leak	√	√	√

6.

Among the four kinds of service usage inefficiency bugs, which kind is the most prevalent?

	Started	Bound	Hybrid	Total
Premature Create	0 (0%)	400 (71.05%)	163 (28.95%)	**563**
Late Destroy	491 (38.18%)	620 (48.21%)	620 (48.21%)	**1286**
Premature Destroy	137 (78.29%)	0 (0%)	38 (21.71%)	**175**
Service Leak	1,720 (71.61%)	392 (16.32%)	290 (12.07%)	**2402**

7.

Among the three types of services, the usage of which type is more prone to inefficiency bugs?

- Started: 2348 out of 4952 (**47,4%**)
- Bound: 1412 out of 2468 (**57,2%**)
- Hybrid: 666 out of 715 (**93,1%**)

Conclusion

- 4 Services Anti-Patterns
- Unneeded Memory Allocation and Energy Inefficiencies
- ServDroid

THANKS!

Any questions?

Douglas Soares – dsl
Jônatas de Oliveira – joc