

Sicherheitsanalyse für Android- Systemdienste auf der Basis von Programm-Slicing

Evaluation eines Tools zur statischen Codeanalyse von
Android-Systemdiensten auf der Grundlage des
WALA-Frameworks



Agenda

- Ziel der Arbeit
- Rechtekonzept von Android
- Einführung in das Programm-Slicing
- Demonstration des Tools
- Test des Tools
- Quellen



Ziel der Arbeit

- Entwicklung eines Tools zum Slicing von Android-Systemdiensten, aka „Android-Slicer“
- Nutzung des WALA-Frameworks von IBM
 - Codeanalyse-Framework für Java-Bytecode
- Zeitgemäße GUI
- Einfache und intuitive Einstellung der Parameter
- Sinnvolle Darstellung und Auswertbarkeit der entstandenen Slices



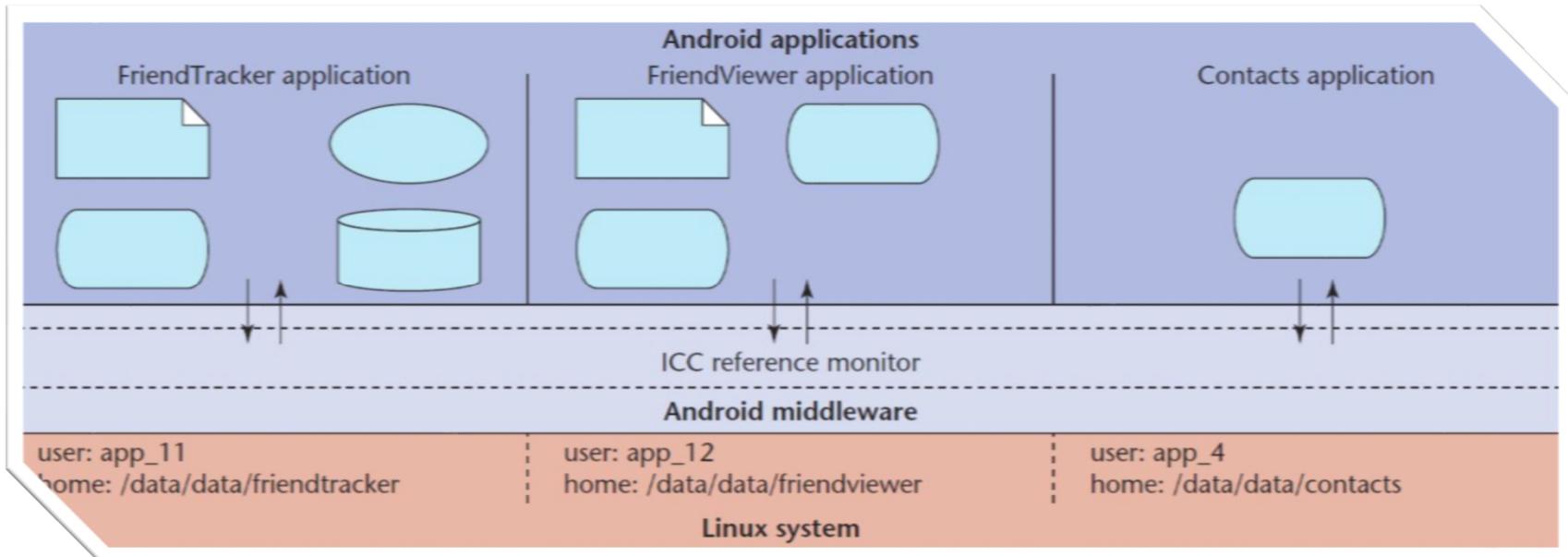
Rechtekonzept von Android

- Deklaratives Rechtekonzept mit Angabe der benötigten Berechtigungen in Manifest-Datei

```
1  <manifest
2      xmlns:android="http://schemas.android.com/apk/res/android"
3      package="com.example.myapp">
4
5      <uses-permission android:name="android.permission.SEND_SMS"/>
6      <uses-permission android:name="android.permission.CAMERA"/>
7      ...
</manifest>
```

Rechtekonzept von Android

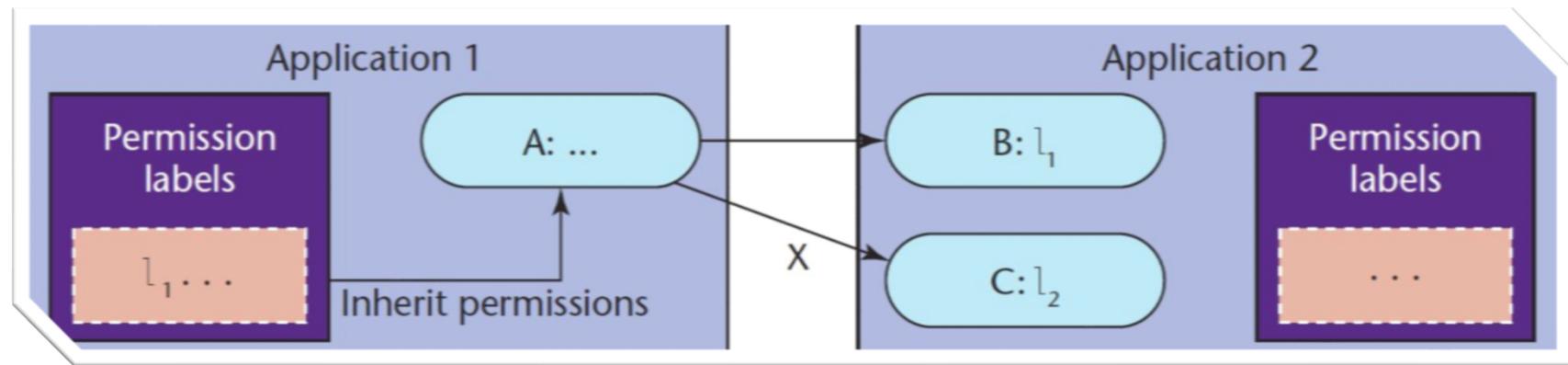
- Rechteprüfung durch Interprozesskommunikation durch Referenz-Monitor



Quelle: Enck et. al. „Understanding Android Security“, S. 53

Rechtekonzept von Android

- Aktionen werden anhand von „Permission-Labels“ erlaubt oder verweigert



Quelle: Enck et. al. „Understanding Android Security“, S. 54



Rechtekonzept von Android

- Systemdienste (oder andere Apps) können Referenz-Monitor durch „Service Hooks“ zur Laufzeit erweitern

```
1  ...
2  package com.android.server;
3  ...
4  class AlarmManagerService extends SystemService {
5  ...
6      private final IBinder mService = new IAlarmManager.Stub() {
7          ...
8          public boolean setTime(long millis){
9              getContext().enforceCallingOrSelfPermission(
10                  "android.permission.SET_TIME",
11                  "setTime");
12              return setTimeImpl(millis);
13          }
14      ...
}
```



Rechtekonzept von Android

- Spezifikation der Interfaces erfolgt in AIDL

```
1  ...
2  package android.app;
3  ...
4  interface IAlarmManager {
5      /** windowLength == 0 means exact; windowLength < 0 means let the OS decide
6          */
7      void set(String callingPackage, int type, long triggerAtTime, long windowLength
8              ,
9              long interval, int flags, Intent operation, IAlarmListener
10             listener,
11             String listenerTag, WorkSource workSource, AlarmManager.
12             AlarmClockInfo alarmClock);
13     boolean setTime(long millis);
14     void setTimeZone(String zone);
15     void remove(Intent operation, IAlarmListener listener);
16     long getNextWakeFromIdleTime();
17     AlarmManager.AlarmClockInfo getNextAlarmClock(int userId);
18     long currentNetworkTimeMillis();
19 }
```



Rechtekonzept von Android

- Beispiele für Service Hooks

Funktion	Beschreibung
checkCallingPermission(String permission)	Prüft, ob der aufrufende IPC-Prozess im Besitz einer bestimmten Berechtigung ist.
checkCallingOrSelfPermission(String permission)	Prüft, ob der aufrufende IPC-Prozess oder der eigene Prozess im Besitz einer bestimmten Berechtigung ist.
checkPermission(String permission, int pid, int uid)	Prüft, ob der Prozess mit einer bestimmten Prozess- und Benutzer-ID im Besitz einer bestimmten Berechtigung ist.
enforceCallingOrSelfPermission(String permission, String message)	Prüft, ob der aufrufende IPC-Prozess oder der eigene Prozess im Besitz einer bestimmten Berechtigung ist und wirft ggf. eine SecurityException.
enforceCallingPermission(String permission, String message)	Prüft, ob der aufrufende IPC-Prozess im Besitz einer bestimmten Berechtigung ist und wirft ggf. eine SecurityException.
enforcePermission(String permission, int pid, int uid, String message)	Prüft, ob der Prozess mit einer bestimmten Prozess- und Benutzer-ID im Besitz einer bestimmten Berechtigung ist und wirft ggf. eine SecurityException.



Programm-Slicing

- Unterscheidung zwischen
 - Backward-Slicing (Was hat diese Anweisung beeinflusst?)
 - Forward-Slicing (Was wird diese Anweisung beeinflussen?)

```
1     sum = 0;
2     mul = 1;
3     a = 1;
4     b = read();
5     while (a <= b) {
6         sum = sum + a;
7         mul = mul * a;
8         a = a + 1;
9     }
10    write(sum);
11    write(mul);
```



Programm-Slicing

- Forward-Slicing

```
1      sum = 0;
2      mul = 1;
3      a = 1;
4      b = read();
5      while (a <= b) {
6          sum = sum + a;
7          mul = mul * a;
8          a = a + 1;
9      }
10     write(sum);
11     write(mul);
```



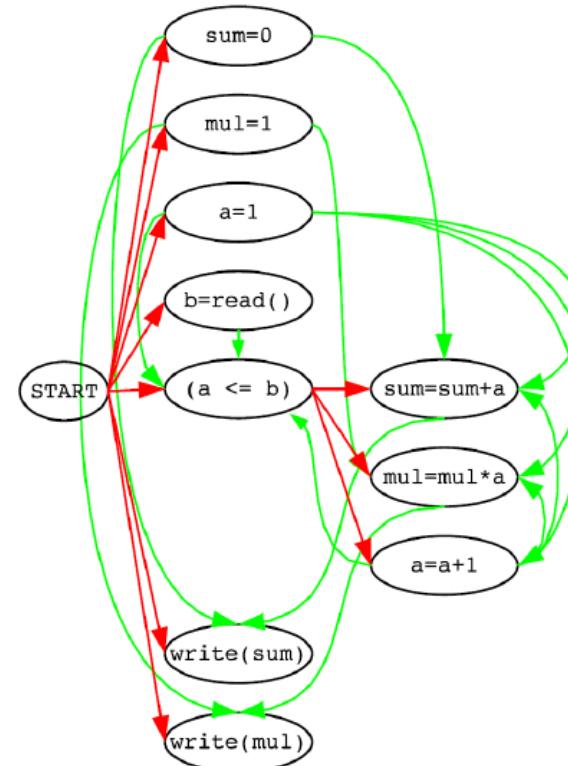
Programm-Slicing

- Backward-Slicing

```
1  sum = 0;
2  mul = 1;
3  a = 1;
4  b = read();
5  while (a <= b) {
6      sum = sum + a;
7      mul = mul * a;
8      a = a + 1;
9  }
10 write(sum);
11 write(mul);
```

Programm-Slicing

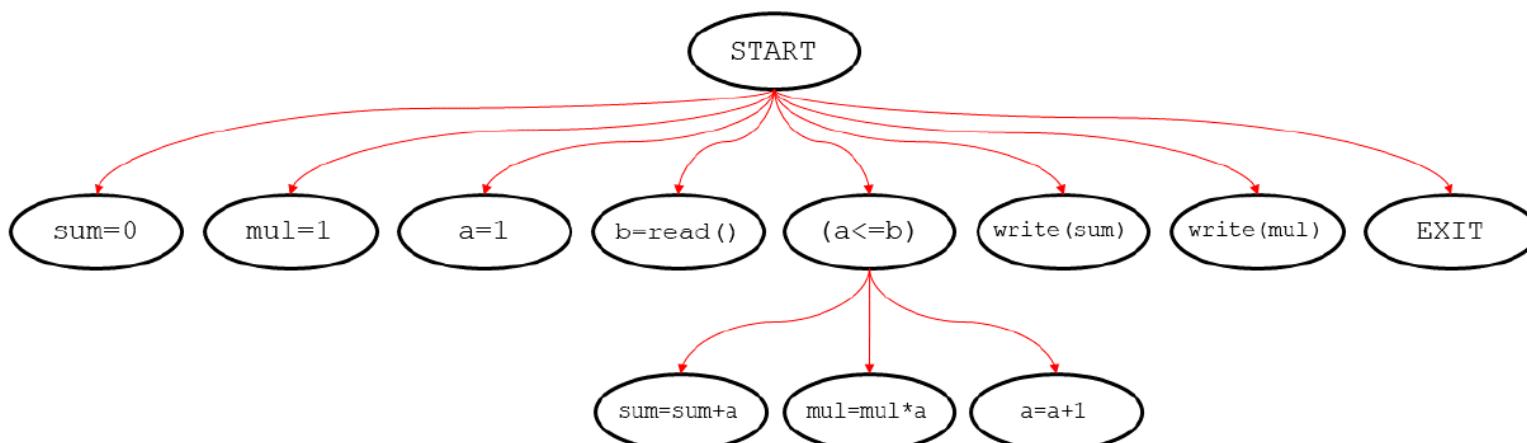
- Intraprozedurales Slicing auf Basis des PDG
 - verknüpft Kontroll- und Datenabhängigkeiten
 - gerichteter Graph
 - Knoten entsprechen Anweisungen und Prädikaten
 - Kanten repräsentieren Abhängigkeiten



Quelle: Robschink, Torsten. „Pfadbedingungen in Abhängigkeitsgraphen und ihre Anwendung in der Softwaresicherheitstechnik“, S. 16

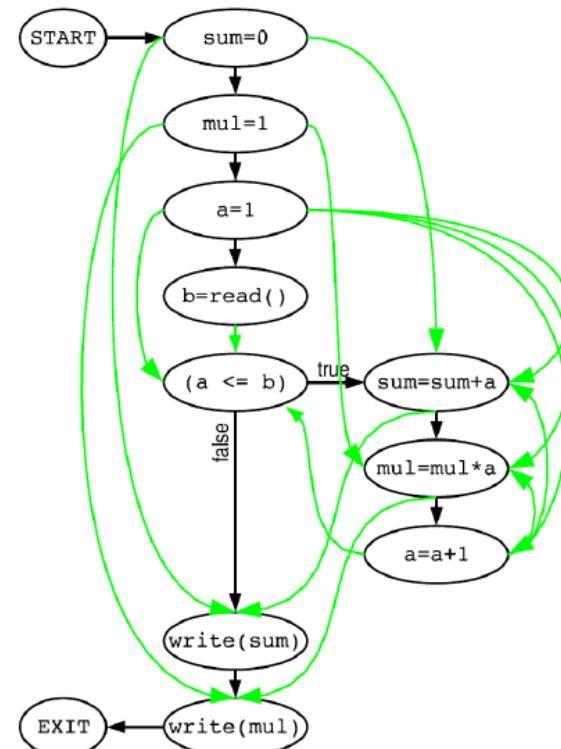
Programm-Slicing

- Kontrollabhängigkeiten werden durch Analyse des Kontrollflusses bestimmt



Programm-Slicing

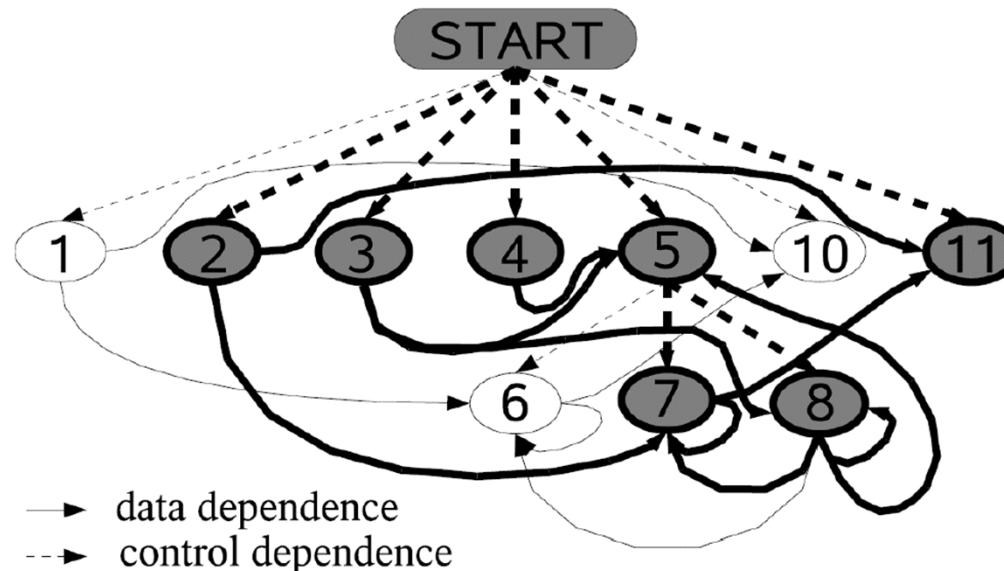
- Datenabhängigkeiten basieren auf Set-Use-Beziehungen
 - Set = Definition einer Variablen
 - Use = Zugriff auf eine Variable
 - dabei erfolgt Berücksichtigung von „Reaching Definitions“



Quelle: Robschink, Torsten. „Pfadbedingungen in Abhängigkeitsgraphen und ihre Anwendung in der Softwaresicherheitstechnik“, S. 15

Programm-Slicing

- Slicing des Programms (bzw. der Methode) durch rückwärtige Traversierung des PDG



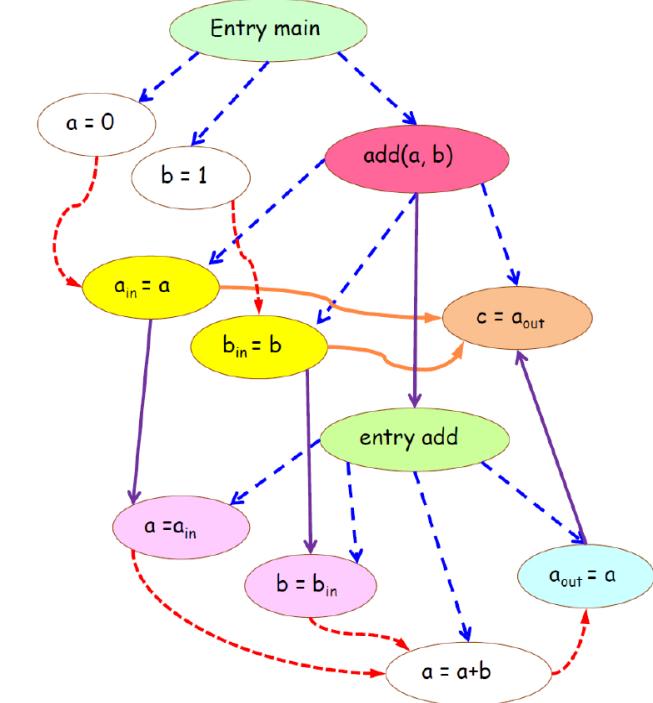
Quelle: Krinke, Jens. „Advanced Slicing of Sequential and Concurrent Programs“, S. 34

Programm-Slicing

- Interprozedurales Slicing auf Basis des SDG
 - verknüpft mehrere PDGs
 - gerichteter Graph
 - zusätzliche Parameter-Knoten
 - zusätzliche Call- und Parameter-Kanten

```
main(){  
    int a, b;  
    a = 0;  
    b = 1;  
    c=add(a, b);  
}  
int add(int a, int b)  
{  
    a = a + b;  
    return a;  
}
```

— Control Dependence
— Data Dependence
— Call, Parameter-in,
Parameter-out
— Summary Edge

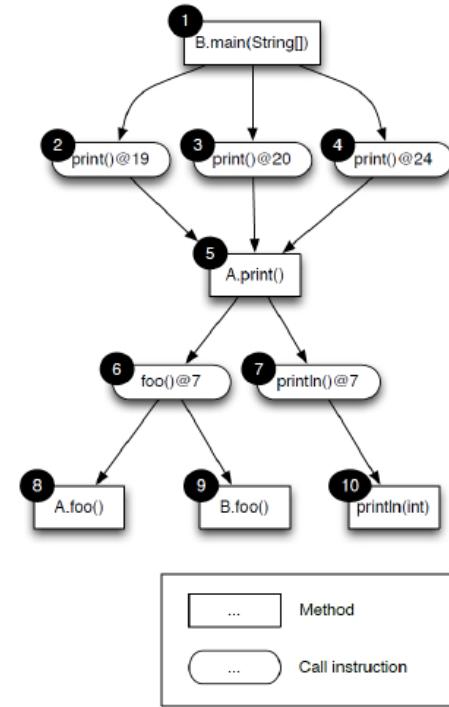


Quelle: Ankur Jain. „Programming Slicing and Its applications“. Folie 23

Programm-Slicing

- Bestimmung der Aufrufe erfolgt auf der Analyse des „Call Graphen“
 - gerichteter, interprozeduraler Kontrollflussgraph
 - stellt „Caller-Callee-Beziehung“ dar

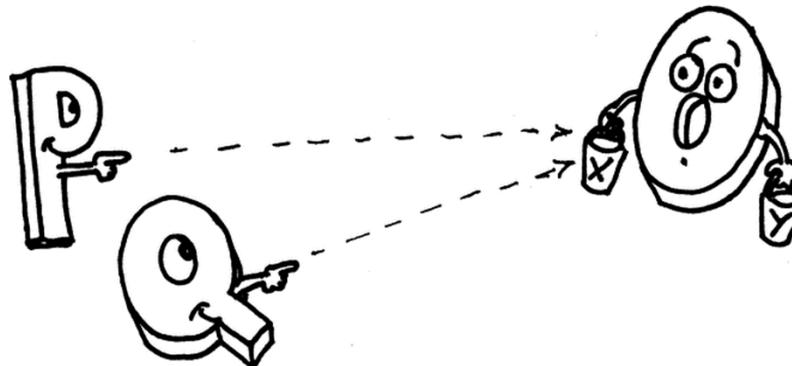
```
1 class A {  
2     int foo() {  
3         return 42;  
4     }  
5     void print() {  
6         println(foo());  
7     }  
8 }  
9  
10 class B extends A {  
11     int foo() {  
12         return 23;  
13     }  
14     ...  
15  
16 void main(String argv[]) {  
17     A a = new A();  
18     A b = new B();  
19     a.print();  
20     b.print();  
21     if (argv[1].equals("B"))  
22         a = b;  
23     a.print();  
24 }  
25 }  
26 }
```





Programm-Slicing

- PDG und Call Graph benötigen für objektorientierte Sprachen Pointer-Analyse



Quelle: Christian Ullnboom. Java Ist Auch Eine Insel – 3.5 Mit Referenzen Arbeiten, Identität Und Gleichheit (online: http://openbook.rheinwerk-verlag.de/javainsel9/javainsel_03_005.htm)

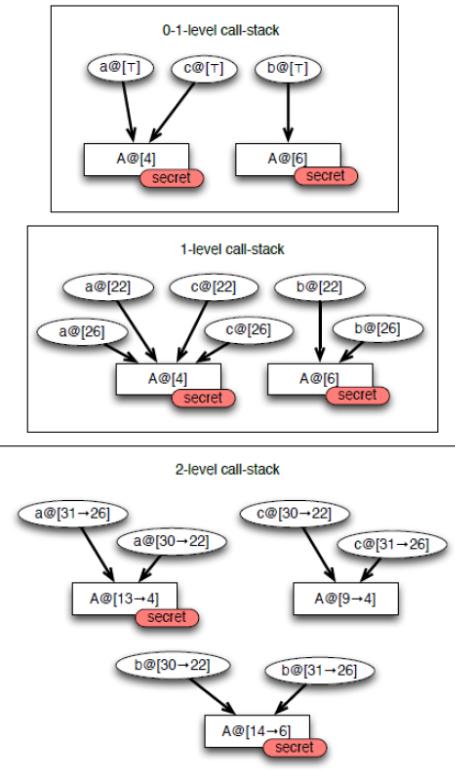
- Auflösung der dynamischen Bindungen (d.h. Bestimmung von konkreten Objektinstanzen)
- Bestimmung von Seiteneffekten bei Methodenaufrufen (benötigt für Parameter-Knoten)

Programm-Slicing

- Pointer-Analysen arbeiten mit Call Stack-basierten Kontexten
 - Referenzkontakte für Variablen
 - Instanzkontakte für Objekte
 - verschiedene „Level“: entsprechen Anzahl an nachverfolgten Methoden auf Call Stack
 - zusätzlich Allozierungs-basierte Ansätze für Container-Objekte

```

1 class A {
2     int data;
3
4     A create1() { return new A(); }
5
6     A create2() { return new A(); }
7
8     A callCreate() {
9         return create1();
10    }
11
12    int ptsStack(int secret) {
13        A a = create1();
14        A b = create2();
15        A c = callCreate();
16        b.data = secret;
17        a.data = secret;
18        return c.data;
19    }
20
21    int call1(int secret) {
22        return ptsStack(secret);
23    }
24
25    int call2(int secret) {
26        return ptsStack(secret);
27    }
28
29    int main(int secret) {
30        return call1(secret)
31            + call2(secret);
32    }
33 }
```



Quelle: Graf, Jürgen. „Information Flow Control with System Dependence Graphs - Improving Modularity, Scalability and Precision for Object Oriented Languages“, S. 71



Demo Android-Slicer



Quellen

- Msridhar und Sjfink. „T.J. Watson Libraries for Analysis (WALA)“, 14. Juni 2015. http://wala.sourceforge.net/wiki/index.php/Main_Page.
- Google Developers. „App Manifest Overview“. Zugegriffen 16. Juli 2019. <https://developer.android.com/guide/topics/manifest/manifest-intro>.
- William Enck, Machigar Ongtang, und Patrick McDaniel. „Understanding Android Security“ IEEE Security & Privacy (2009): 50–57.
- Google Developers. „Context“. Zugegriffen 17. Juli 2019. <https://developer.android.com/reference/android/content/Context>.
- Koschke, Dr Rainer. „Vorlesung Software-Reengineering“: Program Slicing, Universität Bremen, 2010. <https://www.informatik.uni-bremen.de/st/lehre/re10/slicing.pdf>.
- Krinke, Jens. „Advanced Slicing of Sequential and Concurrent Programs“. Universität Passau, 2003. <https://opus4.kobv.de/opus4-uni-passau/frontdoor/index/index/year/2004/docId/38>.
- Ankur Jain. „Programing Slicing and Its applications“. Indian Institute of Technology, Kharagpur, 27. März 2014. <https://www.slideshare.net/AnkurJain89/programing-slicing-static-slice-dynamic-slice-system-dependency-graph>.
- Graf, Jürgen. „Information Flow Control with System Dependence Graphs - Improving Modularity, Scalability and Precision for Object Oriented Languages“. Karlsruher Instituts für Technologie (KIT), 2016. <https://pp.ipd.kit.edu/uploads/publikationen/graf16thesis.pdf>.