Universität Bremen

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

- Ziele der Arbeit
- Systemdienste in Android
- Rechtekonzept und „Service Hooks“
- Einführung in das Programm-Slicing
- Demonstration des Tools
- Test des Tools
- Quellen

Ziele der Arbeit

- Entwicklung eines Tools zum Slicing von Android-Systemdiensten, aka „Android-Slicier“
- 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

Systemdienste in Android

- Zugriff auf Soft- und Hardwarefunktionen erfolgt über Systemdienste
- Insgesamt ca. 50 Dienste
  - Beispiele:

AlarmManagerService: erstellt Alarme, ändert Systemzeit

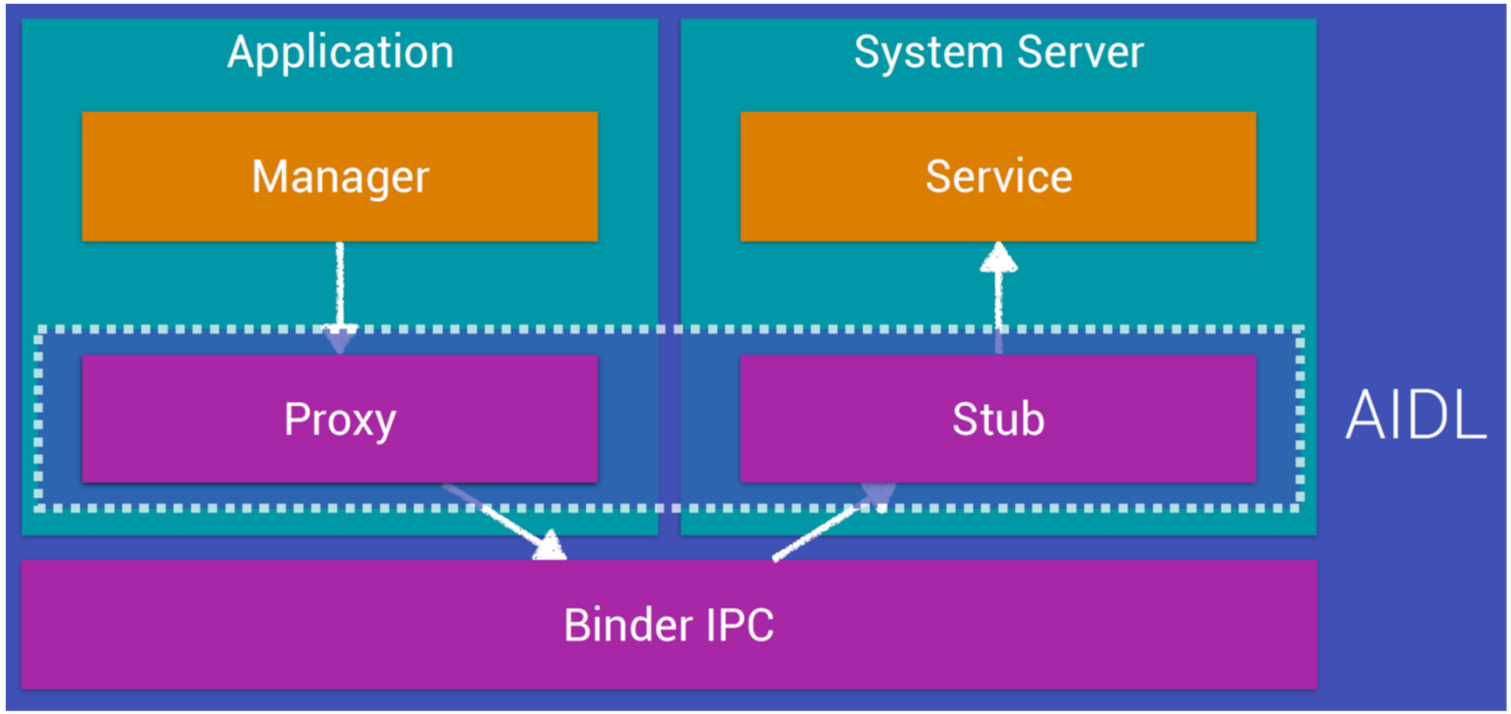
PackageManagerService: installiert und verwaltet Apps

BluetoothManagerService: Zugriff auf Bluetooth-Hardware

PowerManagerService: verwaltet Energiefunktionen (Batteriestatus, Neustarts, etc.)

Systemdienste in Android

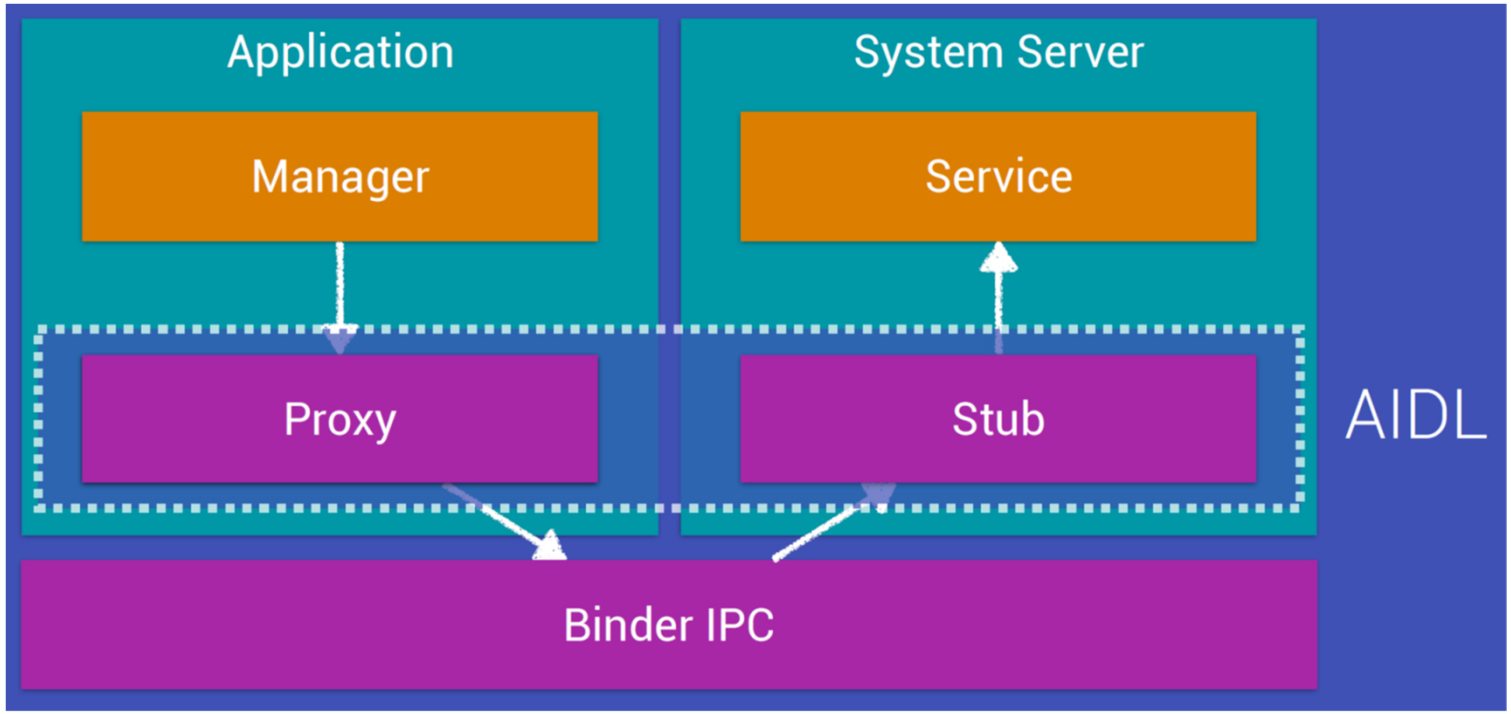
- Apps und Systemdienste werden in unterschiedlichen Prozessen ausgeführt



Quelle: Dave Smith. „Digging Into Android System Services“, Minute 13:55

Systemdienste in Android

- Kommunikation erfolgt daher über Interprozesskommunikation (IPC) mittels „Bindern“



Quelle: Dave Smith. „Digging Into Android System Services“, Minute 13:55

Systemdienste in Android

- Spezifikation der Interfaces für Manager erfolgt in AIDL

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

Rechtekonzept von Android

- Deklaratives Rechtekonzept mit Angabe der Berechtigungen des Aufrufers 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

- Systemdienste (oder andere Apps) können Rechte des Aufrufers durch „Service Hooks“ überprüfen

```
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);
            }
            ...
```

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.

Rechtekonzept von Android

- Fehler bei der Implementierung von Service Hooks führen zu Sicherheitslücken, Beispiel **BluetoothManagerService**:

```
1     public boolean setTrust(String addr, boolean val){
2         if(!BluetoothAdapter.checkBluetoothAddress(addr)){
3             mContext.enforceCallingOrSelfPermission(
4             BLUETOOTH_ADMIN);
5                 return false;
6         }
7         if(!isEnabledInternal()) return false;
8             return setDevicePropertyBooleanNative(...);
9         }
```

Quelle: Tanveer Mustafa und Karsten Sohr. „Understanding the Implemented Access Control Policy of Android System Services with Slicing and Extended Static Checking“, S. 8

Programm-Slicing

- Manuelle Analyse von Systemdiensten ist aufwändig und fehleranfällig (**PackageManagerService** hat fast 25.000 Zeilen Quellcode)
- Programm-Slicing extrahiert relevante Teile und eliminiert den Rest

Programm-Slicing

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

```
1     sum = 0; // was wird diese Anweisung beeinflussen?
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);
1     write(mul); // wie wurde  diese Anweisung beeinflusst?
```

Programm-Slicing

- Forward-Slicing

```
1     sum = 0; // was wird diese Anweisung beeinflussen?
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);
```

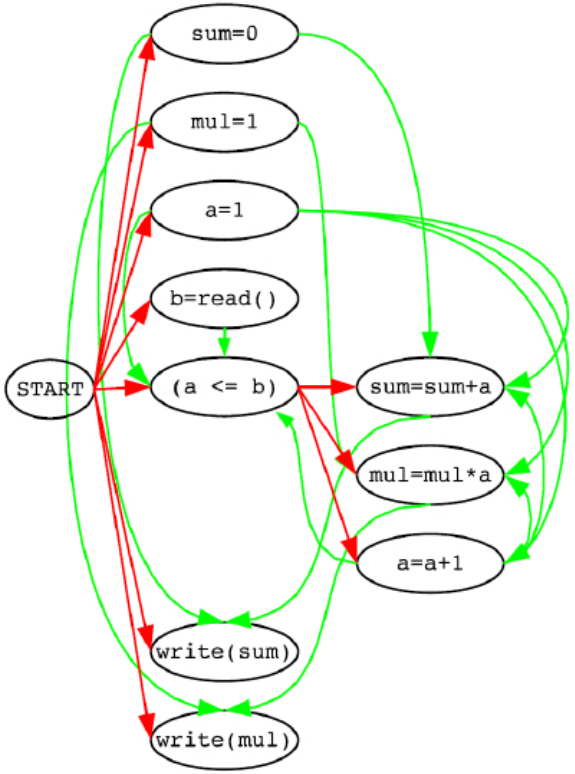
Programm-Slicing

- 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); // wie wurde diese Anweisung beeinflusst?
```

Programm-Slicing

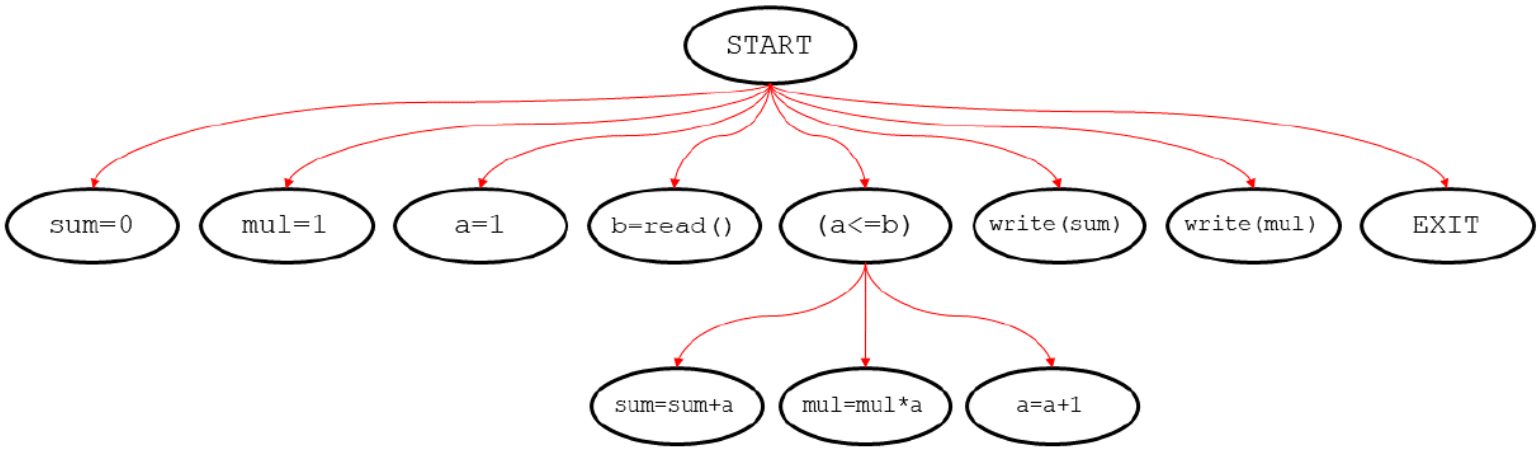
- Intraprozedurales Slicing auf Basis des PDG
  - verknüpft Kontroll- und Datenabhängigkeiten
  - gerichteter Graph
  - Knoten entsprechen Anweisungen und Prädikaten
  - Kanten repräsentieren Abhängigkeiten



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

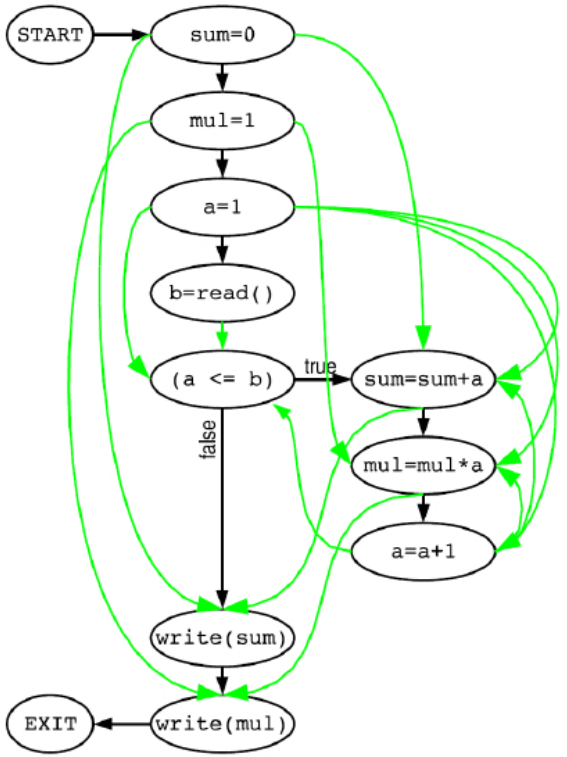
Programm-Slicing

- 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 Defintions“

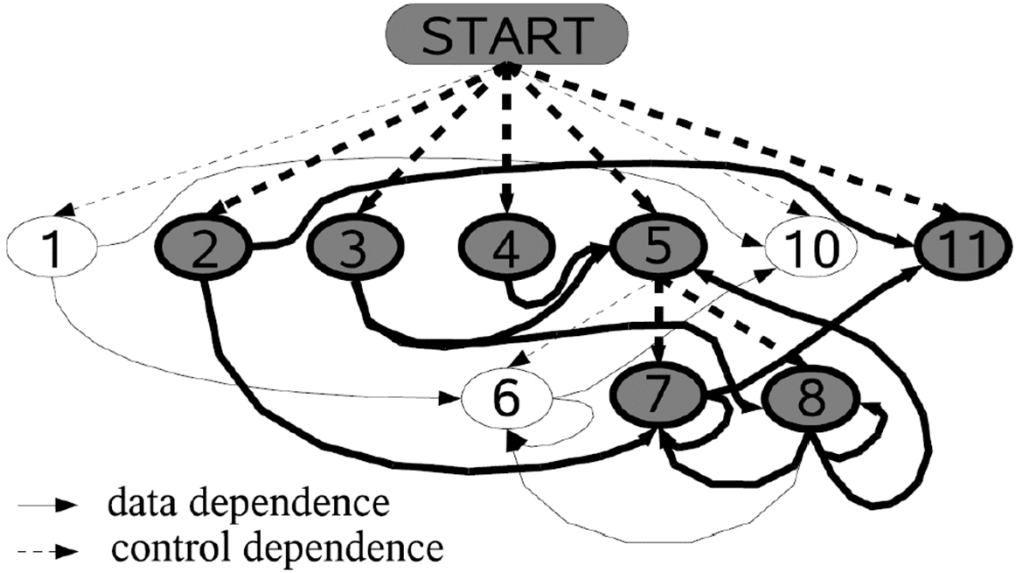


Quelle: Robschink, Torsten. „Pfadbedingungen in Abhängigkeitsgraphen und ihre Anwendung in der Softwaresicherheitstechnik“, S. 15

Programm-Slicing

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

```
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);
```

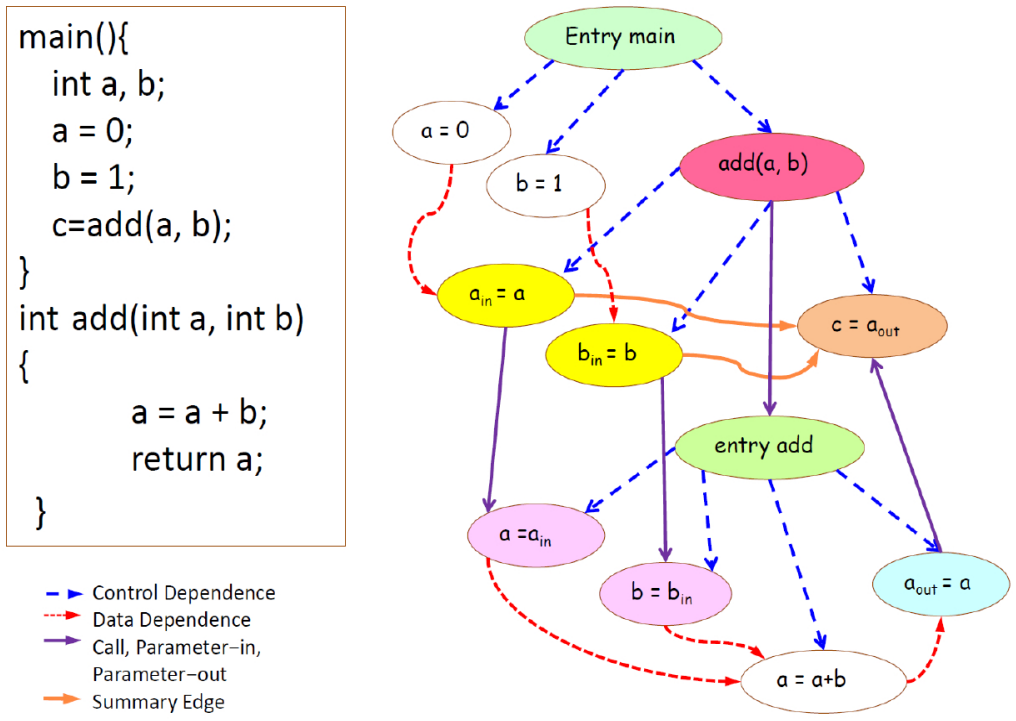


Quelle: Krinke, Jens. „Advanced Slicing of Sequential and Concurrent Programs“, S. 34

Programm-Slicing

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

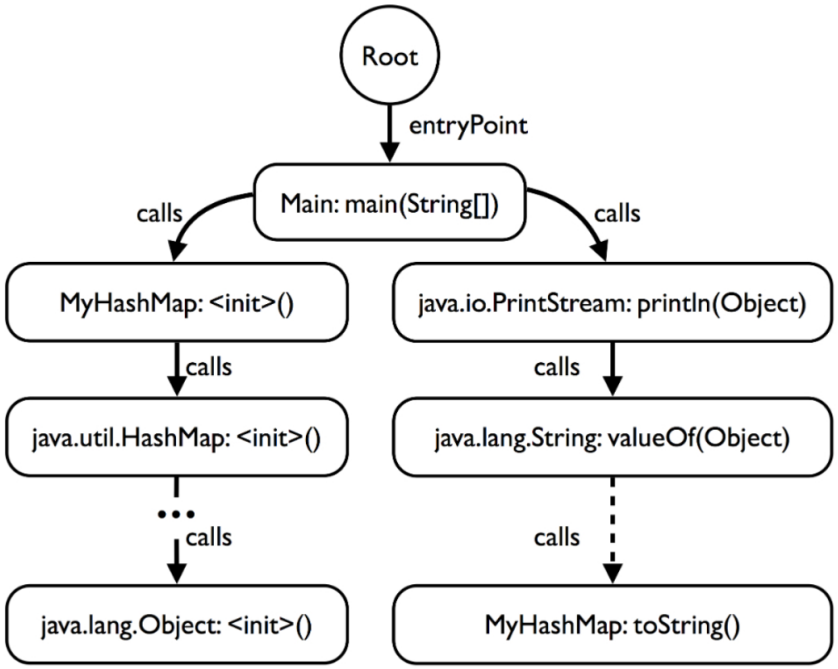
```
main(){
   int a, b;
   a = 0;
   b = 1;
   c=add(a, b);
}
int add(int a, int b)
{
        a = a + b;
        return a;
 }
```



Quelle: Ankur Jain. „Programing Slicing and Its applications". Folie 23

Programm-Slicing

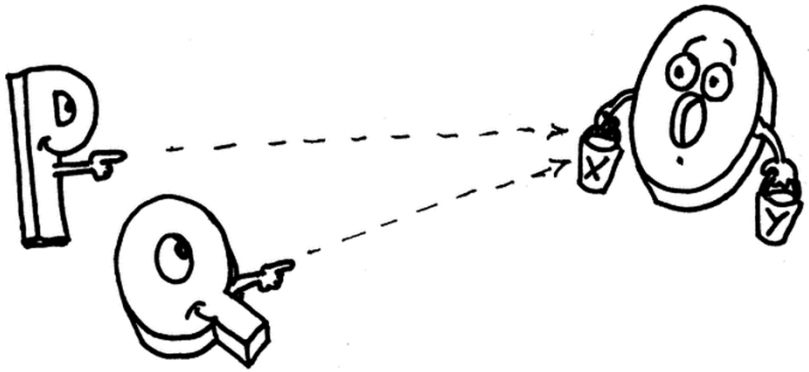
- Bestimmung der Aufrufe erfolgt auf der Analyse des „Call Graphen“
  - gerichteter, interprozeduraler Kontrollflussgraph
  - stellt „Caller-Callee-Beziehung“dar



Quelle: Ali, Karim, und Ondrej Lhotak. „Application-only Call Graph Construction“. S. 5

Programm-Slicing

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



Quelle: Christian Ullenboom. 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 Kontexten zur Unterscheidung der Variablen und Objekte (auch „*k-l-CFA*“ genannt)
  - Kontexte werden über den Call Stack, d.h. anhand des Aufrufers, erstellt*
  - verschiedene „Level“: entsprechen Anzahl an nachverfolgten Methoden auf Call Stack
  - k = Kontext-Level für Variablen
  - l = Kontext-Level für Objekte
  - 0-CFA nutzt lediglich Bezeichner zur Unterscheidung
  - je höher das Level desto höher die Genauigkeit, aber auch der Rechenaufwand
  - 0-1-Level Analyse bietet beste Kombination aus Genauigkeit und Aufwand

*zusätzlich Allozierungs-basierte Ansätze für Container-Objekte

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.
- Dave Smith. „Digging Into Android System Services“. 19. September 2016. https://www.youtube.com/watch?v=M6extgmQQNw.
- 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.