A Debugger for Detecting Security Violations in Android Apps

Guide - Prof. G. Sivakumar and Prof. R.K Shyamasundar

Gopesh Singh Yadav 163050045

Department of Computer Science and Engineering

July 1, 2018

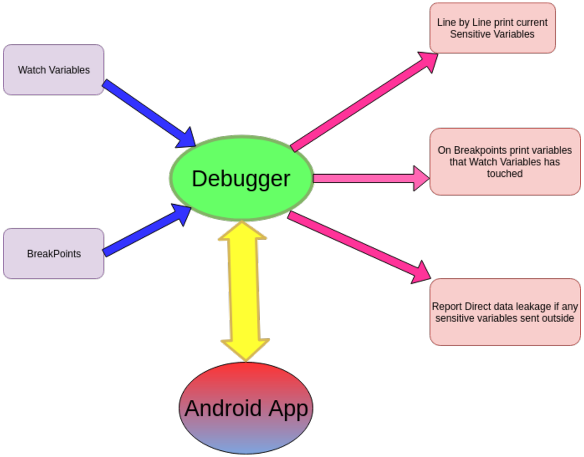
Introduction and Motivation I

- Although there are many dynamic and static analysis tools available for checking security of android app, but there is no debugger available which is designed explicitly for checking security in app's code fragment
- Objective is to build an android debuggers which can be used to debug security in android apps
- The debugger will interact with app running on android device and it will monitor app's behaviour and report if there is any sensitive date leakage

Introduction and Motivation II

- Our debugger allows setting of breakpoints, it monitors app from first breakpoint till last breakpoint
- On each breakpoint prints variables accessed by watch variables starting from first breakpoint
- Prints list of current sensitive variables line by line
- Considers watch variables and variables touching sensitive information e.g, location, contacts etc as sensitive
- Considers any variable touching any sensitive variable as sensitive
- Reports direct data leakage if sensitive variable sent to outside world
- Sensitive sources and sinks list based on APIMonitor [4]

Working Flow of our Debugger


Watch Variables
BreakPoints
Debugger
Android App
Line by Line print current Sensitive Variables
On Breakpoints print variables that Watch Variables has touched
Report Direct data leakage if any sensitive variables sent outside

Implementation

- Used two approaches
  - Using only **Java Debug Interface (JDI)** [1] Library
  - Using both **JDI** and **JavaParser** [3] Libraries

- Maintains current sensitive variables list
- variables touching sensitive variables or sensitve information added into sensitive variables list
- For each variable maintains list of currently touched variables
- Variable A touches variable B then B and variables touched by B included in variables touched by A

Java Platform Debugger Architecture (JPDA) [1]

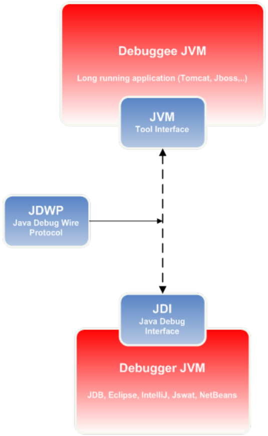


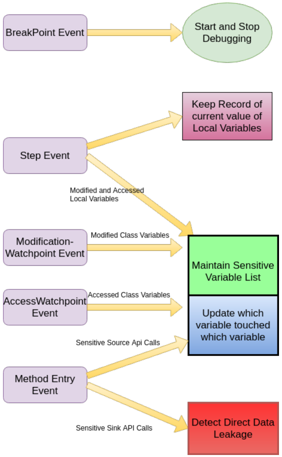
Figure: JPDA Architecture Diagram [5]

Java Debug Interface (JDI)

Event Requests supported by JDI

- ClassPrepare Request
- BreakPoint Request
- FieldModificationWatchpoint Request
- FieldAccessWatchpoint Request
- Step Request
- Method Entry and Exit Request

Debuggger implemented using only JDI Library [1] I


BreakPoint Event
Start and Stop Debugging
Keep Record of current value of Local Variables
Step Event
Modified and Accessed Local Variables
Modification-Watchpoint Event
Modified Class Variables
Maintain Sensitive Variable List
AccessWatchpoint Event
Accessed Class Variables
Update which variable touched which variable
Sensitive Source Api Calls
Method Entry Event
Sensitive Sink API Calls
Detect Direct Data Leakage

Debuggger implemented using only JDI Library [1] II

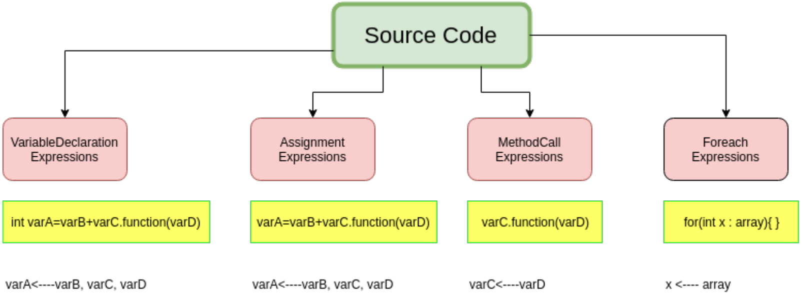
Limitations:-

- No support in JDI to monitor Local Variables
- In expressions like 'varA=varB.functionCall(varC, VarD...)' very slow to find variables used with methods

JavaParser Library [3] I

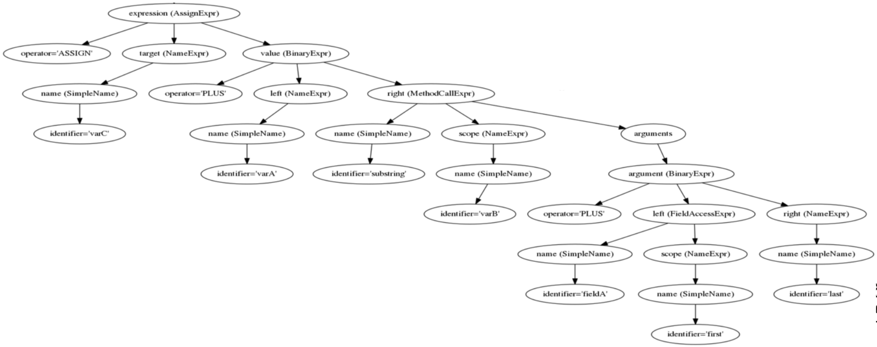
- Parses Java source code into Abstract Syntax Tree (AST)
- Visits AST to find patterns of interest e.g, methods, classes, variables etc
- Allows to manipulate the structure of the source code

Tracking Data Flow in Java Code using JavaParser Library I


Source Code
VariableDeclaration Expressions
Assignment Expressions
MethodCall Expressions
Foreach Expressions
int varA=varB+varC.function(varD)
varA=varB+varC.function(varD)
varC.function(varD)
for(int x : array){ }
varA<----varB, varC, varD
varA<----varB, varC, varD
varC<----varD
x <---- array

Tracking Data Flow in Java Code using JavaParser Library II

AST of assignment expression
varC=varA+varB.substring(first.fieldA+last)



Debugger implemented using both JDI [1] and JavaParser Libraries [3]

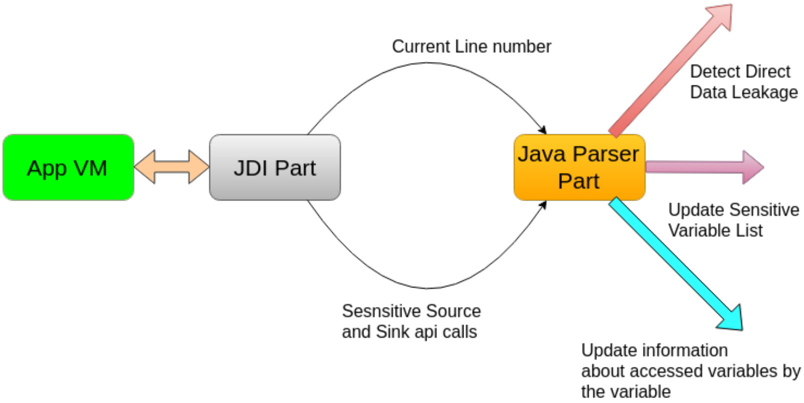
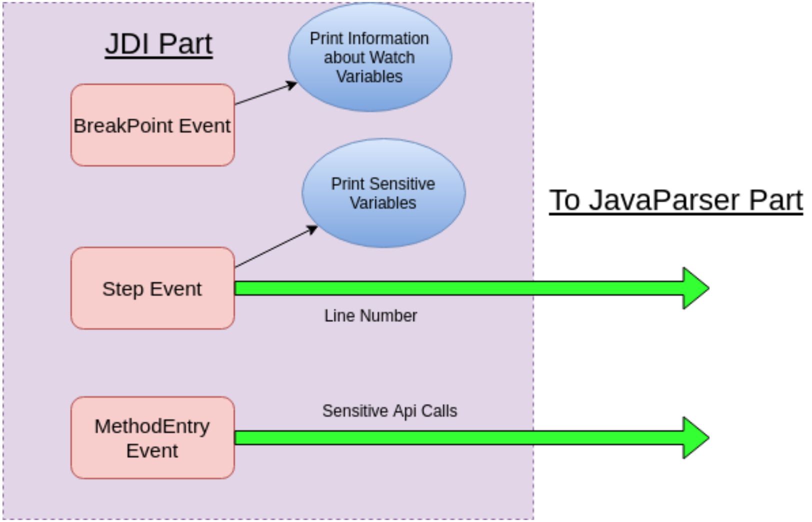

App VM
JDI Part
Java Parser Part
Current Line number
Sesnsitive Source and Sink api calls
Detect Direct Data Leakage
Update Sensitive Variable List
Update information about accessed variables by the variable

Figure: Debugger Implementation

JDI Part


JDI Part
BreakPoint Event
Print Information about Watch Variables
Print Sensitive Variables
Step Event
Line Number
MethodEntry Event
Sensitive Api Calls
To JavaParser Part

JavaParser Part

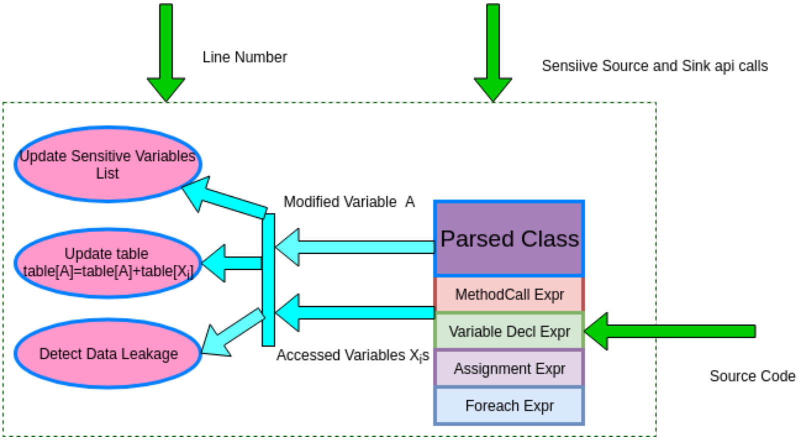
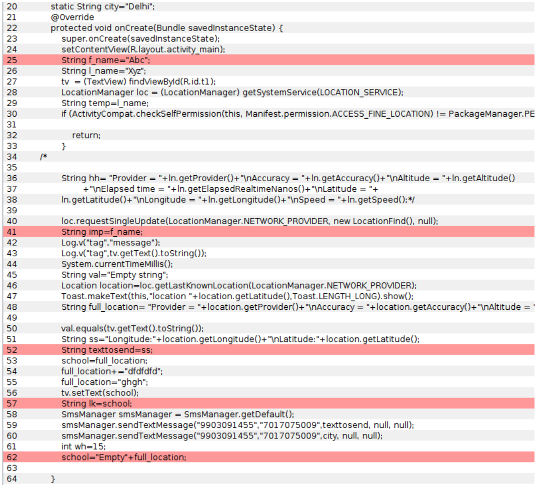

Line Number
Sensiive Source and Sink api calls
Update Sensitive Variables List
Modified Variable A
Parsed Class
Update table table[A]=table[A]+table[X$_i$]
MethodCall Expr
Variable Decl Expr
Detect Data Leakage
Accessed Variables X$_i$s
Assignment Expr
Source Code
Foreach Expr

Figure: JavaParser Part

Comparison of Tools used for Logging and Automating the Debugging Process

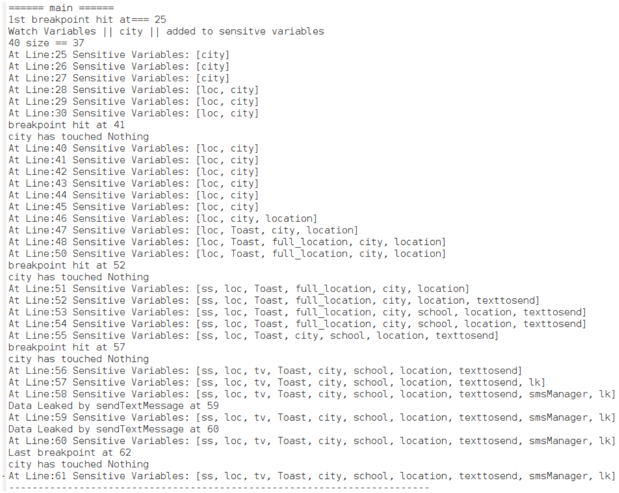
Tool	**Fills form**	**App Instru- -mentation**	**Strategy**	**Supported Android versions**	**Code Coverage**
Android- -ViewClient [6]	Yes	No	Random View & Random Event	All versions	Ok
DroidBot [7]	No	No	Model	All versions	Good
DroidMate [8]	No	Yes	Model	Api 19 & 23	Good
Monkey [9]	No	No	Random	All versions	Less
Utility small Debugger	Manual	No	Manual	All versions	Manual

Experimental Analysis I



```
    static String city="Delhi";
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
        String f_name="Abc";
        String l_name="Xyz";
        tv  = (TextView) findViewById(R.id.t1);
        LocationManager loc = (LocationManager) getSystemService(LOCATION_SERVICE);
        String temp=l_name;
        if (ActivityCompat.checkSelfPermission(this, Manifest.permission.ACCESS_FINE_LOCATION) != PackageManager.PE

            return;
        }
  /*

        String hh= "Provider = "+ln.getProvider()+"\nAccuracy = "+ln.getAccuracy()+"\nAltitude = "+ln.getAltitude()
                +"\nElapsed time = "+ln.getElapsedRealtimeNanos()+"\nLatitude = "+
        ln.getLatitude()+"\nLongitude = "+ln.getLongitude()+"\nSpeed = "+ln.getSpeed();*/

        loc.requestSingleUpdate(LocationManager.NETWORK_PROVIDER, new LocationFind(), null);
        String imp=f_name;
        Log.v("tag","message");
        Log.v("tag",tv.getText().toString());
        System.currentTimeMillis();
        String val="Empty string";
        Location location=loc.getLastKnownLocation(LocationManager.NETWORK_PROVIDER);
        Toast.makeText(this,"location "+location.getLatitude(),Toast.LENGTH_LONG).show();
        String full_location= "Provider = "+location.getProvider()+"\nAccuracy = "+location.getAccuracy()+"\nAltitude = "

        val.equals(tv.getText().toString());
        String ss="Longitude:"+location.getLongitude()+"\nLatitude:"+location.getLatitude();
        String texttosend=ss;
        school=full_location;
        full_location+="dfdfdfd";
        full_location="ghgh";
        tv.setText(school);
        String lk=school;
        SmsManager smsManager = SmsManager.getDefault();
        smsManager.sendTextMessage("9903091455","7017075009",texttosend, null, null);
        smsManager.sendTextMessage("9903091455","7017075009",city, null, null);
        int wh=15;
        school="Empty"+full_location;

    }
```

Experimental Analysis II



```
====== main ======
1st breakpoint hit at=== 25
Watch Variables || city || added to sensitve variables
40 size == 37
At Line:25 Sensitive Variables: [city]
At Line:26 Sensitive Variables: [city]
At Line:27 Sensitive Variables: [city]
At Line:28 Sensitive Variables: [loc, city]
At Line:29 Sensitive Variables: [loc, city]
At Line:30 Sensitive Variables: [loc, city]
breakpoint hit at 41
city has touched Nothing
At Line:40 Sensitive Variables: [loc, city]
At Line:41 Sensitive Variables: [loc, city]
At Line:42 Sensitive Variables: [loc, city]
At Line:43 Sensitive Variables: [loc, city]
At Line:44 Sensitive Variables: [loc, city]
At Line:45 Sensitive Variables: [loc, city]
At Line:46 Sensitive Variables: [loc, city, location]
At Line:47 Sensitive Variables: [loc, Toast, city, location]
At Line:48 Sensitive Variables: [loc, Toast, full_location, city, location]
At Line:50 Sensitive Variables: [loc, Toast, full_location, city, location]
breakpoint hit at 52
city has touched Nothing
At Line:51 Sensitive Variables: [ss, loc, Toast, full_location, city, location]
At Line:52 Sensitive Variables: [ss, loc, Toast, full_location, city, location, texttosend]
At Line:53 Sensitive Variables: [ss, loc, Toast, full_location, city, school, location, texttosend]
At Line:54 Sensitive Variables: [ss, loc, Toast, full_location, city, school, location, texttosend]
At Line:55 Sensitive Variables: [ss, loc, Toast, city, school, location, texttosend]
breakpoint hit at 57
city has touched Nothing
At Line:56 Sensitive Variables: [ss, loc, tv, Toast, city, school, location, texttosend]
At Line:57 Sensitive Variables: [ss, loc, tv, Toast, city, school, location, texttosend, lk]
At Line:58 Sensitive Variables: [ss, loc, tv, Toast, city, school, location, texttosend, smsManager, lk]
Data Leaked by sendTextMessage at 59
At Line:59 Sensitive Variables: [ss, loc, tv, Toast, city, school, location, texttosend, smsManager, lk]
Data Leaked by sendTextMessage at 60
At Line:60 Sensitive Variables: [ss, loc, tv, Toast, city, school, location, texttosend, smsManager, lk]
Last breakpoint at 62
city has touched Nothing
At Line:61 Sensitive Variables: [ss, loc, tv, Toast, city, school, location, texttosend, smsManager, lk]
-------------------------------------------------------------------------
```

Future Work

- Debugging inside called methods
- Resolving method call types to increase performance
- Resolving variables types to handle variables with same names

References I

[1] Java Platform Debugger Architecture (JPDA) https://docs.oracle.com/javase/8/docs/technotes/guides/jpda/architecture.html

[2] Java Platform Debugger Architecture (JPDA) https://docs.oracle.com/javase/8/docs/technotes/guides/jpda/trace.html

[3] JavaParser Library http://javaparser.org/ and https://github.com/javaparser/javaparser

[4] APIMonitor http://javaparser.org/ and https://code.google.com/archive/p/droidbox/wikis/APIMonitor.wiki

References II

[5] Java Platform Debugger Architecture Diagram https://premaseem.files.wordpress.com/2012/12/jpda.png

[6] AndroidViewClient https://github.com/dtmilano/AndroidViewClient

[7] DroidBot https://github.com/honeynet/droidbot

[8] DroidMate https://github.com/konrad-jamrozik/droidmate

[9] Application Exerciser Monkey https://developer.android.com/studio/test/monkey