//if true then for string values, only track jsa resolved strings (so hot spot strings)

public static final boolean ONLY\_TRACK\_JSA\_STRINGS = true;

SourceCallInfoChain.java:

/\*\* Calculate the score for each entry in the call chain \*\*/

public void calculate\_scores() {

score = 0;

if (contents.length == 0) {

API api = API.v();

Set<InfoKind> source = api.getSourceInfoKinds(method);

Set<InfoKind> sink = api.getSinkInfoKinds(method);

if (is\_system (method)) {

if (api.isSafeMethod(method))

score = 0;

else if (api.isSpecMethod(method))

score = 5;

else if (api.isBannedMethod(method))

score = 6;

if (!source.isEmpty())

score += 1;

else if (!sink.isEmpty())

score += 2;

}

return;

}

Score is used to identify whether the source call info chain is a safe method, a spec method, a banned method, or a method with many sources/sinks.

SourceCallChainBuilder:

/\*\* timer so we can timeout after a reasonable amount of time \*\*/

private StopWatch timer = new StopWatch();

/\*\* Timeout (milliseconds). The output will be incomplete \*\*/

private int timeout;

/\*\* just ignore duplicate methods rather than merging call chains \*\*/

private boolean ignore\_dup\_methods = false;

Files and the places that generate the file:

InterAppSourceFlows.gson 🡪 android.droidsafe.analyses.interapp. GenerateInterAppSourceFlows.java

Configurations: createRCFG, produceInterAppFlowsFile (writeinterappflows)

reachable-user-class-methods.txt 🡪 android.droidsafe.analyses.rcfg.RCFG.java

high-level.txt 🡪 android.droidsafe.speclang.SecuritySpecification.java

(number of srcKind -> sinkKind)

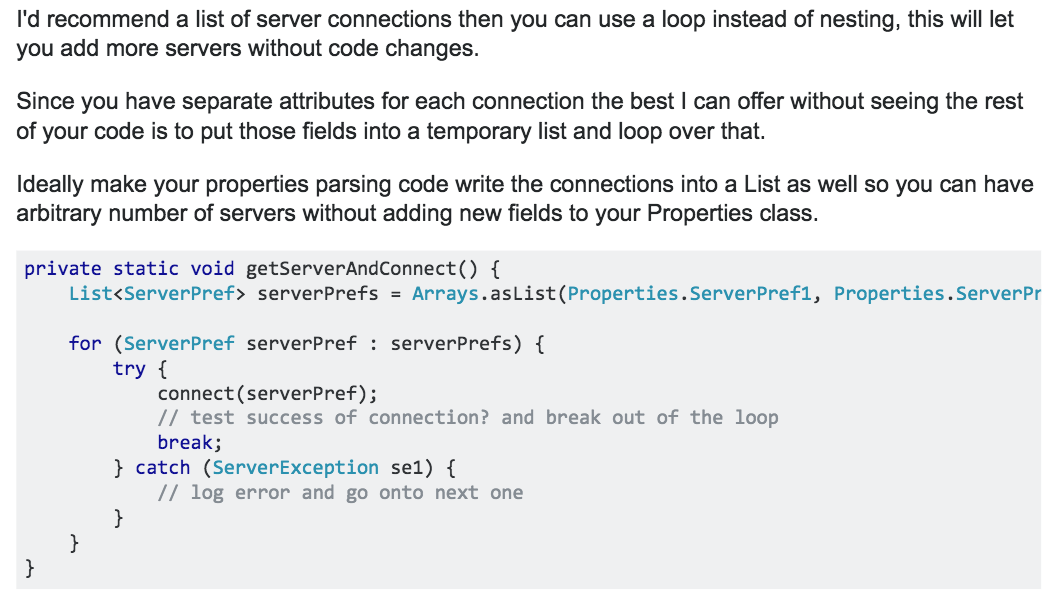
Main.java:

AfterTransform(arg…): Called after one or more transforms to recalculate any underlying analysis. (Running PTA, Finished PTA)

Callback && Fallback

Callback: Your app wants to download a file from some remote computer and then write to to a local disk. The remote computer is the other side of a dial-up modem and a satellite link. The latency and transfer time will be huge and you have other things to do. So, you have a function/method that will write a buffer to disk. You pass a pointer to this method to your network API, together with the remote URI and other stuff. This network call returns 'immediately' and you can do your other stuff. 30 seconds later, the first buffer from the remote computer arrives at the network layer. The network layer then calls the function that you passed during the setup and so the buffer gets written to disk - the network layer has 'called back'. Note that, in this example, the callback would happen on a network layer thread than the originating thread, but that does not matter - the buffer still gets written to the disk.

Fallback: more like a backup mechanism;



addFallbackModeling:

android.droidsafe.transforms.CallBackModeling.java

public void run() {

//go through all user classes and find method that override system methods that are not called

//create a method for each class that calls inherited methods that are not reachable (and not verified)

//for each allocation of one of these classes, call the fallback callback method

findDeadCallbackAndCreateFallbackMethod();

//call the methods that we have created at each allocation

callFallBackMethods();

//check for components that are not created anywhere

findAndCreateUnallocedComponents();

}

android.droidsafe.reports.InfoFlowReportNoRCFG.java: Generate information flow result with RCFG. Only assume that PTA and information flow analysis has completed.

stmt-flows.txt

reachable-app-methods.txt

all-sources.txt (printSensitiveSources())

android.droidsafe.reports.InformationFlowReport.java: Create an information flow report that lists sensitive sinks if a sensitive source flows into a receiver or argument.

info-flow-results.txt (// format is: \*\*|<entry-method>|<sink>/<cat>/line|rx-src|<source\_rx/cat\_list>|arg-src|<src\_arg list)

info-iflow-results.txt (Config.v().implicitFlow) (iflow\_create(SecuritySpecification spec): Create an information flow report that lists sensitive sinks from sensitive sources via implicit flow.)

check-info-flow.txt (Config.v().checkInfoFlow)

android.droidsafe.reports.SensitiveSources.java: (building indicator, writeJSONReports() in Main.java)

sensitive\_sources.json

android.droidsafe.analyses.value.ValueAnalysis.java:

va-results.log&va-errors.log

pta packages:

SPARK: A PTA bridge for the SPARK points to analysis. Right now the analysis is context insensitive.

Config.v().dumpPta 🡪 pta.txt; Config.v().dumpCallGraph 🡪 callgraph.gexf; dumpReachablesAndAllocNodes () 🡪 spark-dump.log; dumpOutdegreesCSV() 🡪 reachables-outdegree.csv; dumpCallGraphReachablesCSV() 🡪 reachables-count.csv

PaddlePTA:

GeoPTA: Config.v().dumpPta 🡪 pta.txt; Config.v().dumpCallGraph 🡪 callgraph.dot

PTAPaper.java & Method.java:

int flowsIntoSinksArgs = 0;

int flowsIntoSinksArgsPrecise = 0;

int flowsIntoSinksMem = 0;

int flowsIntoSinksRec = 0;

android.droidsafe.stats.IntentResolutionStats.java:

icc-resolution-stats.txt

android.droidsafe.stats.FindAPICallsWithNonLocalEffects.java:

non-local-api-methods.txt

android.droidsafe.stats.AveInfoFlowSetSize.java:

info-set-size.txt (totalSet, totalSetsSize)

android.droidsafe.reports.AnalysisReport.java: This class stores data and produces a report for an analyst of any issues encountered during the analysis that are suspicious or could lead to imprecision or inaccuracy in the results.

analysis-warning-report.txt