



THE UNIVERSITY
OF BRITISH COLUMBIA



ReSeSS

The Reliable, Secure, and Sustainable
Software Lab

Dynamic Slicing with Trace-Based Alias Analysis

Khaled Ahmed, Ph.D. candidate, UBC



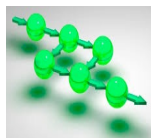
Mobile software

(malware detection, efficient testing, energy-efficiency)



Analysis for microservice-based cloud software

(quality, security, performance, efficient deployment)



Compositional software development

(integration and upgrade management, automated repairs)



Trustworthy AI

(security, reliability, quality assurance, fairness, ethics)

Reliable, Secure, and Sustainable Software Lab (ReSeSS)



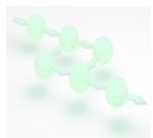
Mobile software

(malware detection, efficient testing, energy-efficiency)



Analysis for microservice-based cloud software

(quality, security, performance, efficient deployment)



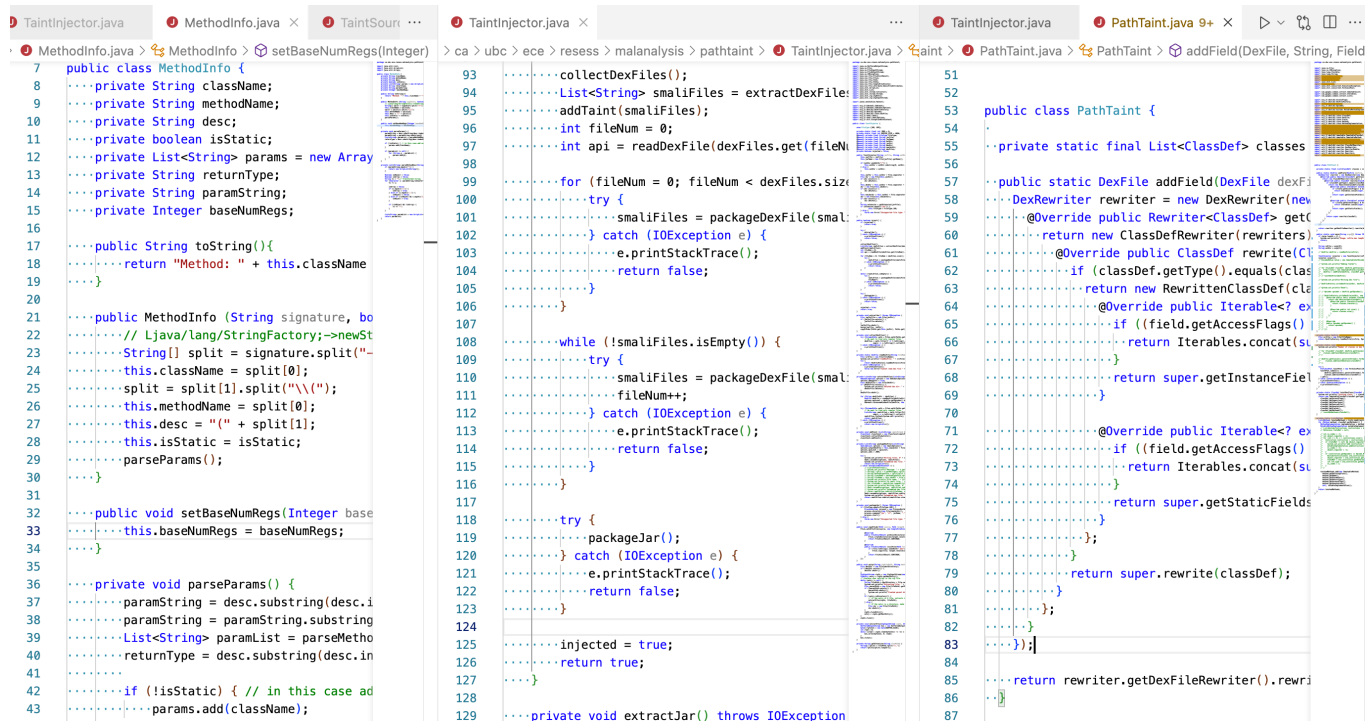
Compositional software development

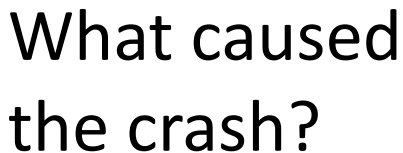
(integration and upgrade management, automated repairs)



Trustworthy AI

(security, reliability, quality assurance, fairness, ethics)







Debugging ...



```
TaintInjector.java  MethodInfo.java  TaintSourc...  TaintInjector.java  PathTaint.java 9+  ...
> ca > ubc > ece > resex > malanalysis > pathtaint > TaintInjector.java > Taint > PathTaint > addField(DexFile, String, Field)

7  public class MethodInfo {
8  ...private String className;
9  ...private String methodName;
10 ...private String desc;
11 ...private boolean isStatic;
12 ...private List<String> params = new Array
13 ...private String returnType;
14 ...private String paramString;
15 ...private Integer baseNumRegs;
16
17 ...public String toString(){
18 ...return "Method: " + this.className
19 ...}
20
21 ...public MethodInfo (String signature, bo
22 ...// Ljava/lang/StringFactory;-newSt
23 ...String[] split = signature.split("-"
24 ...this.className = split[0];
25 ...split = split[1].split("\\(");
26 ...this.methodName = split[0];
27 ...this.desc = "(" + split[1];
28 ...this.isStatic = isStatic;
29 ...parseParams();
30 ...}
31
32 ...public void setBaseNumRegs(Integer base
33 ...this.baseNumRegs = baseNumRegs;
34 ...}
35
36 ...private void parseParams() {
37 ...paramString = desc.substring(desc.i
38 ...paramString = paramString.substring
39 ...List<String> paramList = parseMetho
40 ...returnType = desc.substring(desc.in
41 ...
42 ...if (!isStatic) { // in this case ad
43 ...params.add(className);
44 ...}
45 ...}
46 ...}
47 ...}
48 ...}
49 ...}
50 ...}
51 ...}
52 ...}
53 ...}
54 ...}
55 ...}
56 ...}
57 ...}
58 ...}
59 ...}
60 ...}
61 ...}
62 ...}
63 ...}
64 ...}
65 ...}
66 ...}
67 ...}
68 ...}
69 ...}
70 ...}
71 ...}
72 ...}
73 ...}
74 ...}
75 ...}
76 ...}
77 ...}
78 ...}
79 ...}
80 ...}
81 ...}
82 ...}
83 ...}
84 ...}
85 ...}
86 ...}
87 ...}

93 ...collectDexFiles();
94 ...List<String> smaliFiles = extractDexFile
95 ...addTaint(smaliFiles);
96 ...int fileNum = 0;
97 ...int api = readDexFile(dexFiles.get(fileN
98 ...
99 ...for (fileNum = 0; fileNum < dexFiles.size
100 ...try {
101 ...smaliFiles = packageDexFile(smali
102 ...} catch (IOException e) {
103 ...e.printStackTrace();
104 ...return false;
105 ...}
106 ...}
107
108 ...while (!smaliFiles.isEmpty()) {
109 ...try {
110 ...smaliFiles = packageDexFile(smali
111 ...fileNum++;
112 ...} catch (IOException e) {
113 ...e.printStackTrace();
114 ...return false;
115 ...}
116 ...}
117
118 ...try {
119 ...packageJar();
120 ...} catch (IOException e) {
121 ...e.printStackTrace();
122 ...return false;
123 ...}
124
125 ...injected = true;
126 ...return true;
127 ...}
128
129 ...private void extractJar() throws IOException

51
52
53 public class PathTaint {
54
55 ...private static final List<ClassDef> classes
56
57 ...public static DexFile addField(DexFile dexFi
58 ...DexRewriter rewriter = new DexRewriter(new
59 ...@Override public Rewriter<ClassDef> getCl
60 ...return new ClassDefRewriter(rewriters)
61 ...@Override public ClassDef rewrite(Cl
62 ...if (classDef.getType().equals(class
63 ...return new RewrittenClassDef(class
64 ...@Override public Iterable<? ex
65 ...if ((field.getAccessFlags() &
66 ...return Iterables.concat(su
67 ...}
68 ...return super.getInstanceField
69 ...}
70
71 ...@Override public Iterable<? ex
72 ...if ((field.getAccessFlags() &
73 ...return Iterables.concat(su
74 ...}
75 ...return super.getStaticFields
76 ...}
77 ...};
78 ...return super.rewrite(classDef);
79 ...}
80 ...}
81 ...}
82 ...}
83 ...}
84
85 ...return rewriter.get
86 ...}
87 ...}
```

What caused
the crash?

Do we need
to read the
full code?



Making debugging faster

```
TaintInjector.java x MethodInfo.java x TaintSource ... TaintInjector.java x Taint ... PathTaint.java x PathTaint.java x PathTaint.java x addField(DexFile, String, Field)

7 public class MethodInfo {
8     ...private String className;
9     ...private String methodName;
10    ...private String desc;
11    ...private boolean isStatic;
12    ...private List<String> params = new Array
13    ...private String returnType;
14    ...private String paramString;
15    ...private Integer baseNumRegs;
16
17    ...public String toString(){
18        ...return "Method: " + this.className
19        ...}
20
21    ...public MethodInfo (String signature, bo
22        ...// Ljava/lang/StringFactory;=>newSt
23        ...String[] split = signature.split("-"
24        ...this.className = split[0];
25        ...split = split[1].split("\\(");
26        ...this.methodName = split[0];
27        ...this.desc = "(" + split[1];
28        ...this.isStatic = isStatic;
29        ...parseParams();
30    ...}
31
32    ...public void setBaseNumRegs(Integer base
33        ...this.baseNumRegs = baseNumRegs;
34    ...}
35
36    ...private void parseParams() {
37        ...paramString = desc.substring(desc.i
38        ...paramString = paramString.substring
39        ...List<String> paramList = parseMetho
40        ...returnType = desc.substring(desc.in
41        ...
42        ...if (!isStatic) { // in this case ad
43        ...params.add(className);
44
45    ...collectDexFiles();
46    ...List<String> smaliFiles = extractDexFiles
47    ...addTaint(smaliFiles);
48    ...int fileNum = 0;
49    ...int api = readDexFile(dexFiles.get(fileN
50
51    ...for (fileNum = 0; fileNum < dexFiles.size
52    ...try {
53        ...smaliFiles = packageDexFile(smal
54    ...} catch (IOException e) {
55        ...e.printStackTrace();
56    ...return false;
57    ...}
58
59    ...while (!smaliFiles.isEmpty()) {
60    ...try {
61        ...smaliFiles = packageDexFile(smal
62        ...fileNum++;
63    ...} catch (IOException e) {
64        ...e.printStackTrace();
65    ...return false;
66    ...}
67
68    ...try {
69        ...packageJar();
70    ...} catch (IOException e) {
71        ...e.printStackTrace();
72    ...return false;
73    ...}
74
75    ...injected = true;
76    ...return true;
77    ...}
78
79    ...private void extractJar() throws IOException
80
81    ...public class PathTaint {
82    ...private static final List<ClassDef> classes
83
84    ...public static DexFile addField(DexFile dexF
85    ...DexRewriter rewriter = new DexRewriter(new
86    ...@Override public Rewriter<ClassDef> getCl
87    ...return new ClassDefRewriter(rewriters)
88    ...@Override public ClassDef rewrite(Class
89    ...if (classDef.getType().equals(Class
90    ...return new RewrittenClassDef(class
91    ...@Override public Iterable<?> ex
92    ...if ((field.getAccessFlags() & 0x000000
93    ...return Iterables.concat(su
94
95    ...return super.getInstanceFields()
96
97    ...}
98
99    ...@Override public Iterable<?> ex
100    ...if ((field.getAccessFlags() & 0x000000
101    ...return Iterables.concat(su
102
103    ...return super.getStaticFields()
104
105    ...}
106
107    ...return super.rewrite(classDef);
108    ...}
109
110    ...}
111
112    ...return rewriter.getDexFileRewriter().rewr
113    ...}
```



Making debugging faster

```
TaintInjector.java x MethodInfo.java x TaintSource.java x TaintInjector.java x PathTaint.java x PathTaint.java x PathTaint.java x addField(DexFile, String, Field)

MethodInfo.java x MethodInfo > setBaseNumRegs(Integer)
7 public class MethodInfo {
8     ...private String className;
9     ...private String methodName;
10    ...private String desc;
11    ...private boolean isStatic;
12    ...private List<String> params = new Array
13    ...private String returnType;
14    ...private String paramString;
15    ...private Integer baseNumRegs;
16
17    ...public String toString(){
18        ...return "Method: " + this.className
19    ...}
20
21    ...public MethodInfo (String signature, bo
22        ...// Ljava/lang/StringFactory;=>newSt
23        ...String[] split = signature.split("-"
24        ...this.className = split[0];
25        ...split = split[1].split("\\(");
26        ...this.methodName = split[0];
27        ...this.desc = "(" + split[1];
28        ...this.isStatic = isStatic;
29        ...parseParams();
30    ...}
31
32    ...public void setBaseNumRegs(Integer base
33        ...this.baseNumRegs = baseNumRegs;
34    ...}
35
36    ...private void parseParams() {
37        ...paramString = desc.substring(desc.i
38        ...paramString = paramString.substring
39        ...List<String> paramList = parseMetho
40        ...returnType = desc.substring(desc.in
41        ...
42        ...if (!isStatic) { // in this case ad
43        ...params.add(className);
44    ...}
45
46    ...}
47
48    ...}
49
50    ...}
51
52    ...}
53
54    ...}
55
56    ...}
57
58    ...}
59
60    ...}
61
62    ...}
63
64    ...}
65
66    ...}
67
68    ...}
69
70    ...}
71
72    ...}
73
74    ...}
75
76    ...}
77
78    ...}
79
80    ...}
81
82    ...}
83
84    ...}
85
86    ...}
87
88    ...}
89
90    ...}
91
92    ...}
93
94    ...}
95
96    ...}
97
98    ...}
99
100   ...}
101   ...}
102   ...}
103   ...}
104   ...}
105   ...}
106   ...}
107   ...}
108   ...}
109   ...}
110   ...}
111   ...}
112   ...}
113   ...}
114   ...}
115   ...}
116   ...}
117   ...}
118   ...}
119   ...}
120   ...}
121   ...}
122   ...}
123   ...}
124   ...}
125   ...}
126   ...}
127   ...}
128   ...}
129   ...}
130   ...}
131   ...}
132   ...}
133   ...}
134   ...}
135   ...}
136   ...}
137   ...}
138   ...}
139   ...}
140   ...}
141   ...}
142   ...}
143   ...}
144   ...}
145   ...}
146   ...}
147   ...}
148   ...}
149   ...}
150   ...}
151   ...}
152   ...}
153   ...}
154   ...}
155   ...}
156   ...}
157   ...}
158   ...}
159   ...}
160   ...}
161   ...}
162   ...}
163   ...}
164   ...}
165   ...}
166   ...}
167   ...}
168   ...}
169   ...}
170   ...}
171   ...}
172   ...}
173   ...}
174   ...}
175   ...}
176   ...}
177   ...}
178   ...}
179   ...}
180   ...}
181   ...}
182   ...}
183   ...}
184   ...}
185   ...}
186   ...}
187   ...}
188   ...}
189   ...}
190   ...}
191   ...}
192   ...}
193   ...}
194   ...}
195   ...}
196   ...}
197   ...}
198   ...}
199   ...}
200   ...}
201   ...}
202   ...}
203   ...}
204   ...}
205   ...}
206   ...}
207   ...}
208   ...}
209   ...}
210   ...}
211   ...}
212   ...}
213   ...}
214   ...}
215   ...}
216   ...}
217   ...}
218   ...}
219   ...}
220   ...}
221   ...}
222   ...}
223   ...}
224   ...}
225   ...}
226   ...}
227   ...}
228   ...}
229   ...}
230   ...}
231   ...}
232   ...}
233   ...}
234   ...}
235   ...}
236   ...}
237   ...}
238   ...}
239   ...}
240   ...}
241   ...}
242   ...}
243   ...}
244   ...}
245   ...}
246   ...}
247   ...}
248   ...}
249   ...}
250   ...}
251   ...}
252   ...}
253   ...}
254   ...}
255   ...}
256   ...}
257   ...}
258   ...}
259   ...}
260   ...}
261   ...}
262   ...}
263   ...}
264   ...}
265   ...}
266   ...}
267   ...}
268   ...}
269   ...}
270   ...}
271   ...}
272   ...}
273   ...}
274   ...}
275   ...}
276   ...}
277   ...}
278   ...}
279   ...}
280   ...}
281   ...}
282   ...}
283   ...}
284   ...}
285   ...}
286   ...}
287   ...}
288   ...}
289   ...}
290   ...}
291   ...}
292   ...}
293   ...}
294   ...}
295   ...}
296   ...}
297   ...}
298   ...}
299   ...}
300   ...}
301   ...}
302   ...}
303   ...}
304   ...}
305   ...}
306   ...}
307   ...}
308   ...}
309   ...}
310   ...}
311   ...}
312   ...}
313   ...}
314   ...}
315   ...}
316   ...}
317   ...}
318   ...}
319   ...}
320   ...}
321   ...}
322   ...}
323   ...}
324   ...}
325   ...}
326   ...}
327   ...}
328   ...}
329   ...}
330   ...}
331   ...}
332   ...}
333   ...}
334   ...}
335   ...}
336   ...}
337   ...}
338   ...}
339   ...}
340   ...}
341   ...}
342   ...}
343   ...}
344   ...}
345   ...}
346   ...}
347   ...}
348   ...}
349   ...}
350   ...}
351   ...}
352   ...}
353   ...}
354   ...}
355   ...}
356   ...}
357   ...}
358   ...}
359   ...}
360   ...}
361   ...}
362   ...}
363   ...}
364   ...}
365   ...}
366   ...}
367   ...}
368   ...}
369   ...}
370   ...}
371   ...}
372   ...}
373   ...}
374   ...}
375   ...}
376   ...}
377   ...}
378   ...}
379   ...}
380   ...}
381   ...}
382   ...}
383   ...}
384   ...}
385   ...}
386   ...}
387   ...}
388   ...}
389   ...}
390   ...}
391   ...}
392   ...}
393   ...}
394   ...}
395   ...}
396   ...}
397   ...}
398   ...}
399   ...}
400   ...}
401   ...}
402   ...}
403   ...}
404   ...}
405   ...}
406   ...}
407   ...}
408   ...}
409   ...}
410   ...}
411   ...}
412   ...}
413   ...}
414   ...}
415   ...}
416   ...}
417   ...}
418   ...}
419   ...}
420   ...}
421   ...}
422   ...}
423   ...}
424   ...}
425   ...}
426   ...}
427   ...}
428   ...}
429   ...}
430   ...}
431   ...}
432   ...}
433   ...}
434   ...}
435   ...}
436   ...}
437   ...}
438   ...}
439   ...}
440   ...}
441   ...}
442   ...}
443   ...}
444   ...}
445   ...}
446   ...}
447   ...}
448   ...}
449   ...}
450   ...}
451   ...}
452   ...}
453   ...}
454   ...}
455   ...}
456   ...}
457   ...}
458   ...}
459   ...}
460   ...}
461   ...}
462   ...}
463   ...}
464   ...}
465   ...}
466   ...}
467   ...}
468   ...}
469   ...}
470   ...}
471   ...}
472   ...}
473   ...}
474   ...}
475   ...}
476   ...}
477   ...}
478   ...}
479   ...}
480   ...}
481   ...}
482   ...}
483   ...}
484   ...}
485   ...}
486   ...}
487   ...}
488   ...}
489   ...}
490   ...}
491   ...}
492   ...}
493   ...}
494   ...}
495   ...}
496   ...}
497   ...}
498   ...}
499   ...}
500   ...}
501   ...}
502   ...}
503   ...}
504   ...}
505   ...}
506   ...}
507   ...}
508   ...}
509   ...}
510   ...}
511   ...}
512   ...}
513   ...}
514   ...}
515   ...}
516   ...}
517   ...}
518   ...}
519   ...}
520   ...}
521   ...}
522   ...}
523   ...}
524   ...}
525   ...}
526   ...}
527   ...}
528   ...}
529   ...}
530   ...}
531   ...}
532   ...}
533   ...}
534   ...}
535   ...}
536   ...}
537   ...}
538   ...}
539   ...}
540   ...}
541   ...}
542   ...}
543   ...}
544   ...}
545   ...}
546   ...}
547   ...}
548   ...}
549   ...}
550   ...}
551   ...}
552   ...}
553   ...}
554   ...}
555   ...}
556   ...}
557   ...}
558   ...}
559   ...}
560   ...}
561   ...}
562   ...}
563   ...}
564   ...}
565   ...}
566   ...}
567   ...}
568   ...}
569   ...}
570   ...}
571   ...}
572   ...}
573   ...}
574   ...}
575   ...}
576   ...}
577   ...}
578   ...}
579   ...}
580   ...}
581   ...}
582   ...}

```



Making debugging faster



```
TaintInjector.java x MethodInfo.java x TaintSour... TaintInjector.java x PathTaint.java 9+ x
MethodInfo.java > MethodInfo > setBaseNumRegs(Integer)
7 public class MethodInfo {
8     private String className;
9     private String methodName;
10    private String desc;
11    private boolean isStatic;
12    private List<String> params = new Array
13    private String returnType;
14    private String paramString;
15    private Integer baseNumRegs;
16
17    public String toString(){
18        return "Method: " + this.className
19    }
20
21    public MethodInfo (String signature, bo
22        // Ljava/lang/StringFactory;->newSt
23        String[] split = signature.split("-"
24        this.className = split[0];
25        split = split[1].split("\\(");
26        this.methodName = split[0];
27        this.desc = "(" + split[1];
28        this.isStatic = isStatic;
29        parseParams();
30    }
31
32    public void setBaseNumRegs(Integer base
33        this.baseNumRegs = baseNumRegs;
34    }
35
36    private void parseParams() {
37        paramString = desc.substring(desc.i
38        paramString = paramString.substring
39        List<String> paramList = parseMetho
40        returnType = desc.substring(desc.in
41    }
42    if (!isStatic) { // in this case ad
43        params.add(className);
44    }
45
46    TaintInjector.java > ca > ubc > ece > resexp > malanalysis > pathtaint > TaintInjector.java > Taint > PathTaint.java > PathTaint > addField(DexFile, String, Field
47
48    public class PathTaint {
49
50    private static final List<ClassDef> classes
51
52    public static DexFile addField(DexFile dexF
53    DexRewriter rewriter = new DexRewriter(new
54    @Override public Rewriter<ClassDef> get
55    return new ClassDefRewriter(rewriters)
56    @Override public ClassDef rewrite(Cl
57    if (classDef.getType().equals(c
58    return new RewrittenClassDef(c
59    @Override public Iterable<? ex
60    if ((field.getAccessFlags()
61    return Iterables.concat(s
62    }
63    return super.getInstanceField
64    }
65    @Override public Iterable<? ex
66    if ((field.getAccessFlags()
67    return Iterables.concat(s
68    }
69    return super.getStaticFields
70    }
71    }
72    @Override public Iterable<? ex
73    if ((field.getAccessFlags()
74    return Iterables.concat(s
75    }
76    return super.getStaticFields
77    }
78    }
79    return super.rewrite(classDef);
80    }
81    }
82    }
83    }
84    }
85    return rewriter.getDexFileRewriter().rewr
86    }
87    }
```



Highlight statements that affect the crash



Highlight statements that affect the crash

Program slicing

Program slicing



```
x = input();  
  
y = input();  
  
if (x > 0)  
    z = x + y;  
    w = x;  
  
else  
    z = x - y;  
    w = y;  
  
print(z);  
return w;
```

Program slicing



```
x = input();
```

```
y = input();
```

```
if (x > 0)
```

```
    z = x + y;
```

```
    w = x;
```

```
else
```

```
    z = x - y;
```

```
    w = y;
```

```
print(z);
```

```
return w;
```

Dataflow



Program slicing



```
x = input();
```

```
y = input();
```

```
if (x > 0)
```

```
    z = x + y;
```

```
    w = x;
```

```
else
```

```
    z = x - y;
```

```
    w = y;
```

```
print(z);
```

```
return w;
```

Dataflow



Program slicing

```
x = input();  
y = input();  
  
if (x > 0)  
    z = x + y;  
    w = x;  
  
else  
    z = x - y;  
    w = y;  
  
print(z);  
return w;
```

Dataflow



Program slicing

```
x = input();
```

```
y = input();
```

```
if (x > 0)
```

```
    z = x + y;
```

```
    w = x;
```

```
else
```

```
    z = x - y;
```

```
    w = y;
```

```
print(z);
```

```
return w;
```

Dataflow



Program slicing

```
x = input();
```

```
y = input();
```

```
if (x > 0)
```

```
    z = x + y;
```

```
    w = x;
```

```
else
```

```
    z = x - y;
```

```
    w = y;
```

```
print(z);
```

```
return w;
```

Dataflow
dependence



Program slicing

```
x = input();
```

```
y = input();
```

```
if (x > 0)
```

```
    z = x + y;
```

```
    w = x;
```

```
else
```

```
    z = x - y;
```

```
    w = y;
```

```
print(z);
```

```
return w;
```

Dataflow
dependence



Control
dependence



Program slicing

```
x = input();
```

```
y = input();
```

```
if (x > 0)
```

```
    z = x + y;
```

```
    w = x;
```

```
else
```

```
    z = x - y;
```

```
    w = y;
```

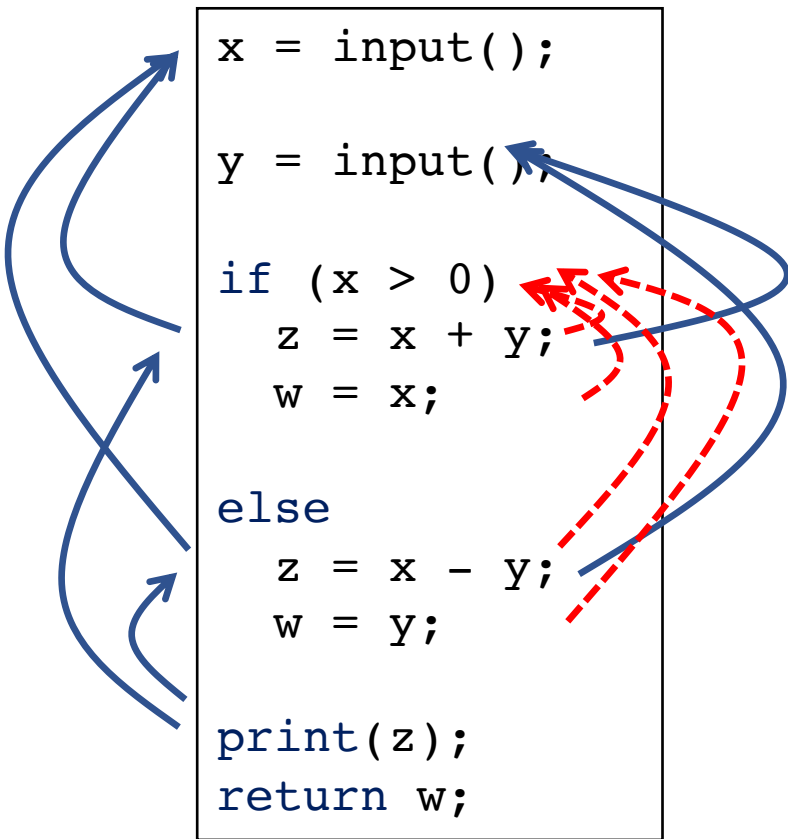
```
print(z);
```

```
return w;
```

Dataflow
dependence

Control
dependence

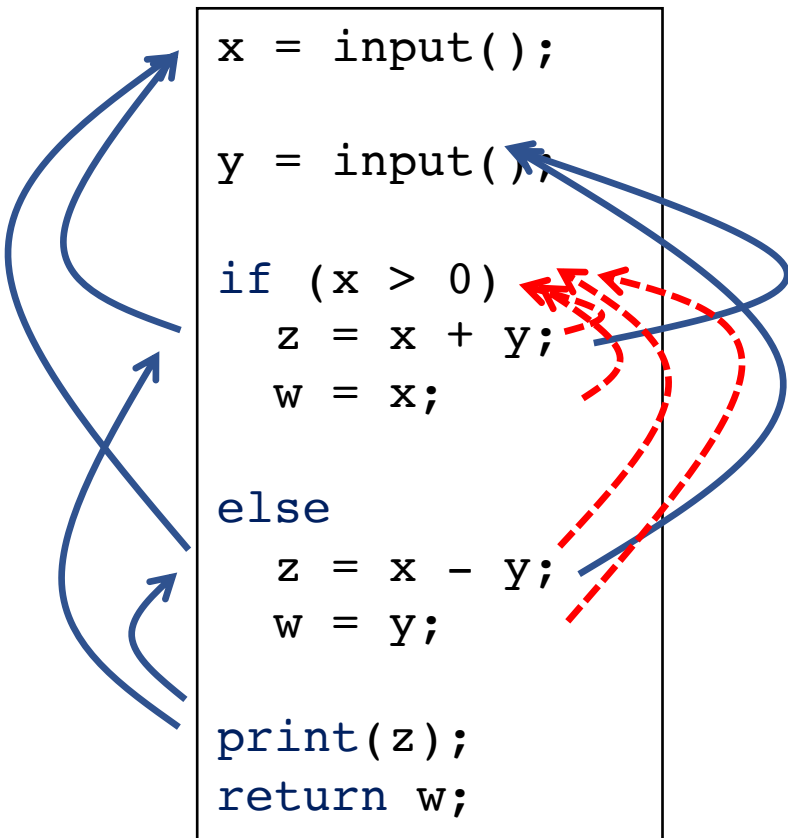
Program slicing



Dataflow
dependence 

Control
dependence 

Program slicing

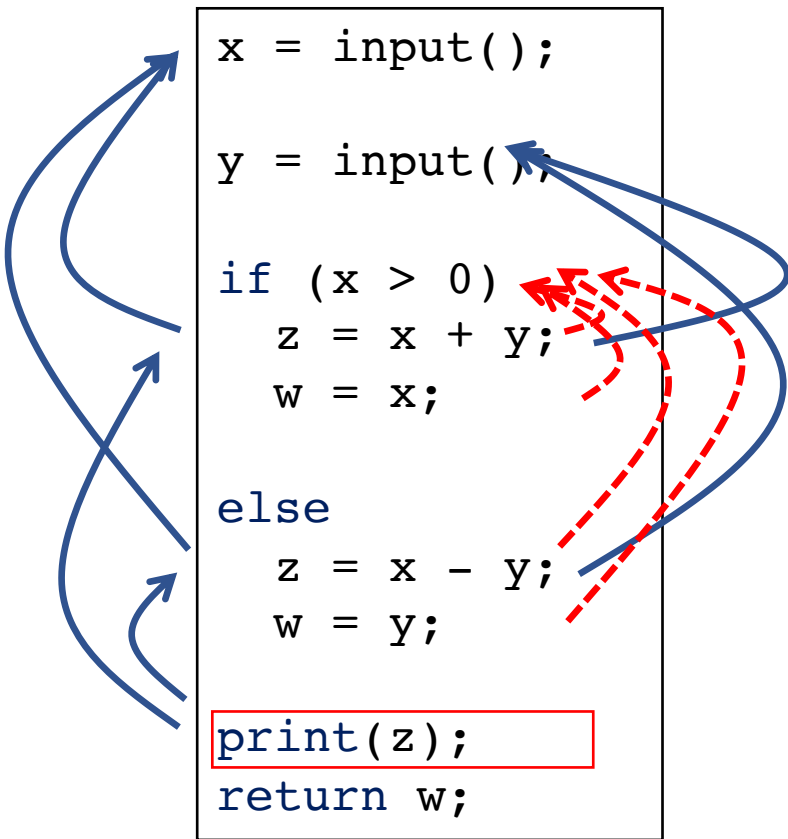


Dataflow
dependence

Control
dependence

Slicing
criterion

Program slicing

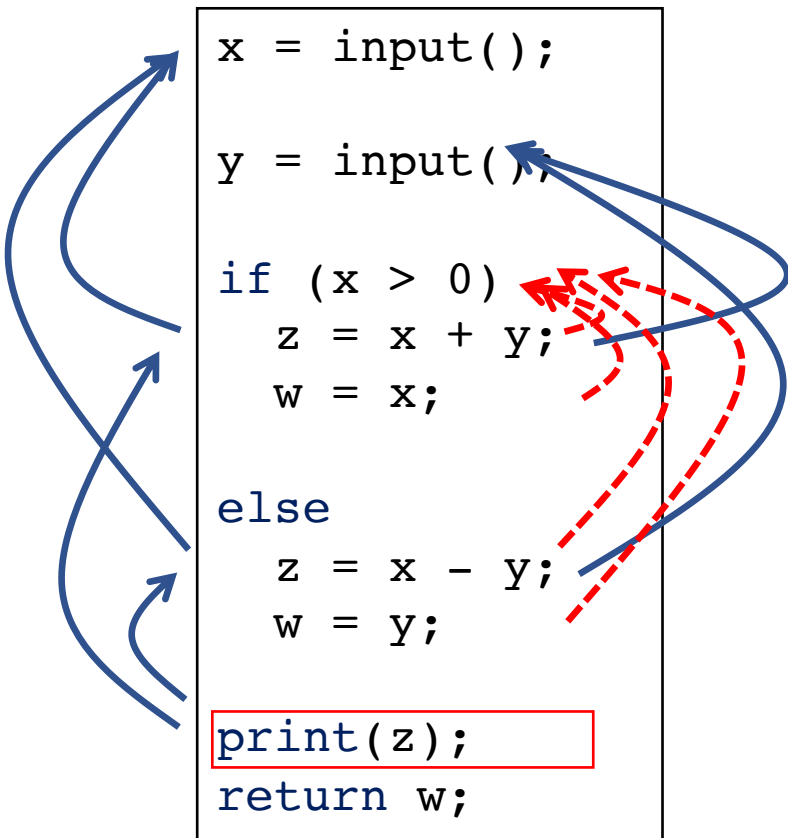


Dataflow
dependence 

Control
dependence 

Slicing
criterion 

Program slicing



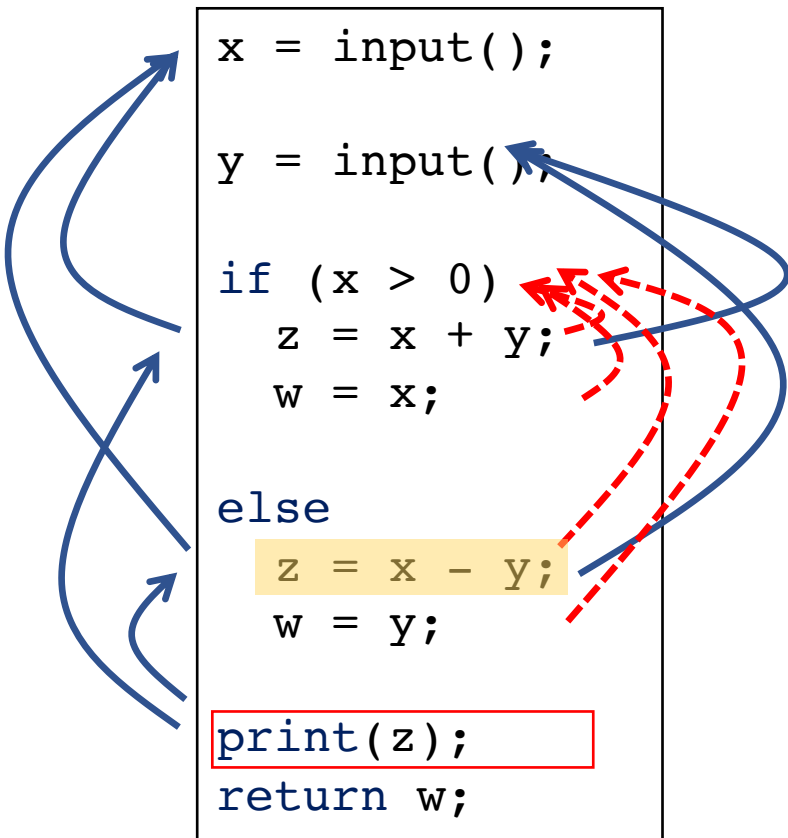
Dataflow
dependence 

Control
dependence 

Slicing
criterion 

Slice 

Program slicing



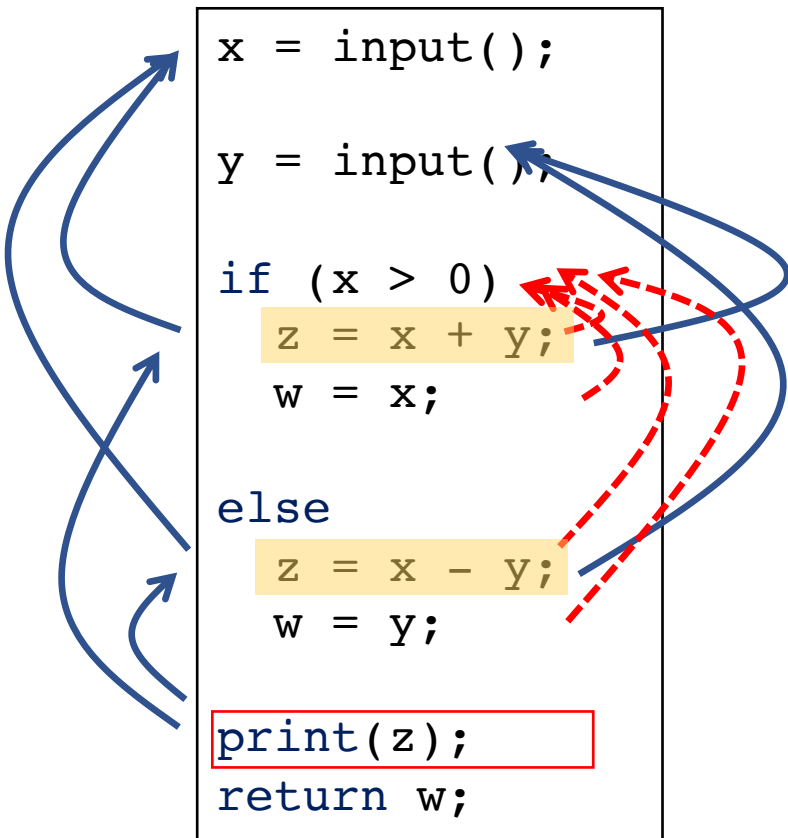
Dataflow
dependence 

Control
dependence 

Slicing
criterion 

Slice 

Program slicing



Dataflow
dependence 

Control
dependence 

Slicing
criterion 

Slice 

Program slicing

```

x = input();
y = input();
if (x > 0)
    z = x + y;
    w = x;
else
    z = x - y;
    w = y;
print(z);
return w;

```

Dataflow
dependence



Control
dependence



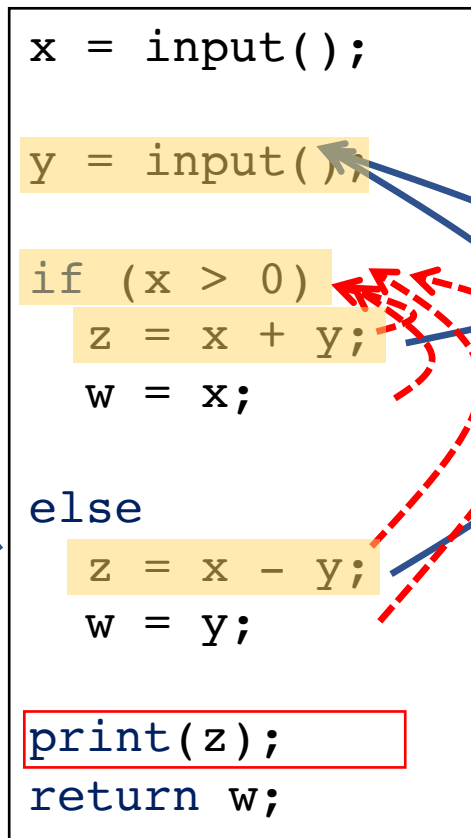
Slicing
criterion



Slice



Program slicing



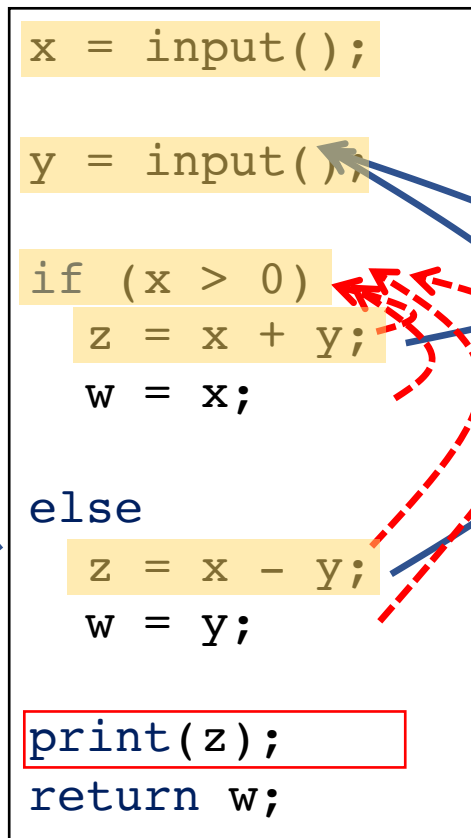
Dataflow
dependence

Control
dependence

Slicing
criterion

Slice

Program slicing



Dataflow dependence 

Control dependence 

Slicing criterion 

Slice 

Static slicing:

Analyze the program without executing it

```
x = input();
```

```
y = input();
```

```
if (x > 0)
```

```
    z = x + y;
```

```
    w = x;
```

```
else
```

```
    z = x - y;
```

```
    w = y;
```

```
print(z);
```

```
return w;
```

Pros

- All execution paths
- No runtime overhead

Cons

- Includes infeasible executions
- Cannot analyze dynamically-loaded code
- Scalability for complex code

Dynamic slicing:

Analyze specific executions of the program

```
x = input();
```

```
y = input();
```

```
if (x > 0)
```

```
    z = x + y;
```

```
    w = x;
```

```
else
```

```
    z = x - y;
```

```
    w = y;
```

```
print(z);
```

```
return w;
```

Pros

- Very accurate for the executed code
- Can analyze dynamically loaded code

Cons

- Triggered paths only
- Runtime overhead

Usefulness of slicing



```
x = input();
```

```
y = input();
```

```
if (x > 0)
```

```
    z = x + y;
```

```
    w = x;
```

```
else
```

```
    z = x - y;
```

```
    w = y;
```

```
print(z);
```

```
return w;
```

Usefulness of slicing

- Aids manual analysis

```
x = input();
```

```
y = input();
```

```
if (x > 0)
```

```
    z = x + y;
```

```
    w = x;
```

```
else
```

```
    z = x - y;
```

```
    w = y;
```

```
print(z);
```

```
return w;
```

Usefulness of slicing

```
x = input();
```

```
y = input();
```

```
if (x > 0)
```

```
    z = x + y;
```

```
    w = x;
```

```
else
```

```
    z = x - y;
```

```
    w = y;
```

```
print(z);
```

```
return w;
```

- Aids manual analysis
- Core of many automated techniques
 - Fault localization
 - Malware detection
 - Refactoring
 - ...

Usefulness of slicing

```
x = input();
```

```
y = input();
```

```
if (x > 0)
```

```
    z = x + y;
```

```
    w = x;
```

```
else
```

```
    z = x - y;
```

```
    w = y;
```

```
print(z);
```

```
return w;
```

- Aids manual analysis
- Core of many automated techniques
 - Fault localization
 - Malware detection
 - Refactoring
 - ...

In the rest of this talk:
we focus on dynamic slicing



Big challenge: dataflow analysis



```
1  jane.age = 15
2  john      = jane
3  john.age  = 25
4  res = jane.age
```



Big challenge: dataflow analysis



```
1  jane.age = 15
```

```
2  john      = jane
```

```
3  john.age = 25
```

```
4  res = jane.age      use
```

Big challenge: dataflow analysis



1 `jane.age = 15`

2 `john = jane`

3 `john.age = 25`

4 `res = jane.age`



use

Big challenge: dataflow analysis



```
1  jane.age = 15
2  john      = jane
3  john.age  = 25
4  res = jane.age
```

definition?

use



Big challenge: dataflow analysis



1 jane.age = 15

2 john = jane

3 john.age = 25

4 res = jane.age



definition?



use

Big challenge: dataflow analysis



1 `jane.age = 15`

2 `john = jane`

3 `john.age = 25`

4 `res = jane.age`



definition?



use

Needs alias analysis



Dynamic dataflow analysis



```
1  jane.age = 15
2  john      = jane
3  john.age  = 25
4  res = jane.age
```



Dynamic dataflow analysis



```
1  jane.age = 15
   print([jane.age])
2  john      = jane
3  john.age = 25
   print([john.age])
4  res = jane.age
   print([jane.age])
```



Dynamic dataflow analysis



1	<code>jane.age = 15</code> <code>print([jane.age])</code>	<code>[jane.age] = 0xA</code>
2	<code>john = jane</code>	
3	<code>john.age = 25</code> <code>print([john.age])</code>	<code>[john.age] = 0xA</code>
4	<code>res = jane.age</code> <code>print([jane.age])</code>	<code>[jane.age] = 0xA</code>

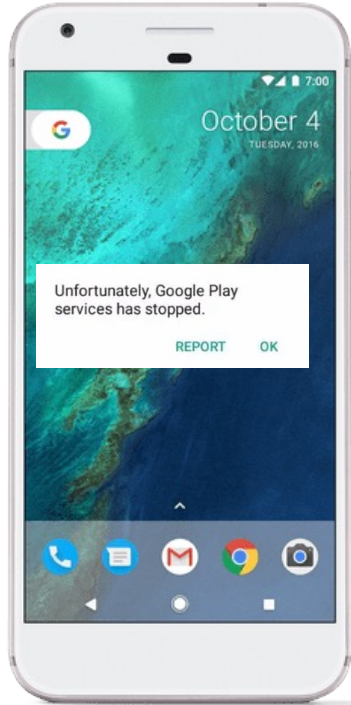
Dynamic dataflow analysis



1	<code>jane.age = 15</code>	<code>[jane.age] = 0xA</code>
	<code>print([jane.age])</code>	
2	<code>john = jane</code>	
3	<code>john.age = 25</code>	<code>[john.age] = 0xA</code>
	<code>print([john.age])</code>	
4	<code>res = jane.age</code>	<code>[jane.age] = 0xA</code>
	<code>print([jane.age])</code>	

Instrumentation (even a very smart one)
→ High overhead

High overhead is bad



On a mobile device:
system will kill slow apps



Efficient dataflow analysis



1	<code>jane.age = 15</code> <code>print([jane.age])</code>	<code>[jane.age] = 0xA</code>
2	<code>john = jane</code>	
3	<code>john.age = 25</code> <code>print([john.age])</code>	<code>[john.age] = 0xA</code>
4	<code>res = jane.age</code> <code>print([jane.age])</code>	<code>[jane.age] = 0xA</code>

Efficient dataflow analysis

```

1  jane.age = 15
   print([jane.age])
2  john     = jane
3  john.age = 25
   print([john.age])
4  res = jane.age
   print([jane.age])

```

```

[jane.age] 0xA
[john.age] = 0xA
[jane.age] 0xA

```



Don't record addresses



*Efficient dynamic slicing for mobile**



* Khaled Ahmed, Mieszko Lis, and Julia Rubin. [MANDOLINE: Dynamic Slicing of Android Applications with Trace-Based Alias Analysis](#). ICST, **Distinguished Paper Award**, 2021



*Efficient dynamic slicing for mobile**



1. Light instrumentation → only basic blocks, not every statement, no fields

* Khaled Ahmed, Mieszko Lis, and Julia Rubin. [MANDOLINE: Dynamic Slicing of Android Applications with Trace-Based Alias Analysis](#). ICST, **Distinguished Paper Award**, 2021



*Efficient dynamic slicing for mobile**



1. Light instrumentation → only basic blocks, not every statement, no fields
2. Post-execution analysis of the trace

* Khaled Ahmed, Mieszko Lis, and Julia Rubin. [MANDOLINE: Dynamic Slicing of Android Applications with Trace-Based Alias Analysis](#). ICST, **Distinguished Paper Award**, 2021



*Efficient dynamic slicing for mobile**



1. Light instrumentation → only basic blocks, not every statement, no fields
2. Post-execution analysis of the trace
 - Recovers dataflows

* Khaled Ahmed, Mieszko Lis, and Julia Rubin. [MANDOLINE: Dynamic Slicing of Android Applications with Trace-Based Alias Analysis](#). ICST, **Distinguished Paper Award**, 2021



*Efficient dynamic slicing for mobile**



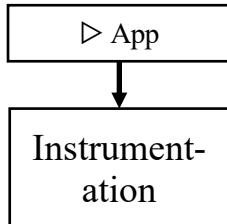
1. Light instrumentation → only basic blocks, not every statement, no fields
2. Post-execution analysis of the trace
 - Recovers dataflows
 - Static, but analyzes one execution path only

* Khaled Ahmed, Mieszko Lis, and Julia Rubin. [MANDOLINE: Dynamic Slicing of Android Applications with Trace-Based Alias Analysis](#). ICST, **Distinguished Paper Award**, 2021

*Efficient dynamic slicing for mobile**



1. Light instrumentation → only basic blocks, not every statement, no fields
2. Post-execution analysis of the trace
 - Recovers dataflows
 - Static, but analyzes one execution path only

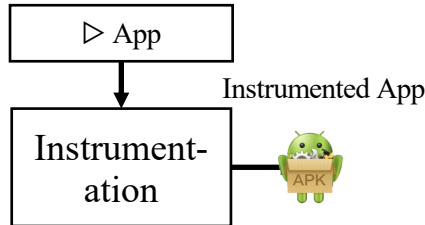


* Khaled Ahmed, Mieszko Lis, and Julia Rubin. [MANDOLINE: Dynamic Slicing of Android Applications with Trace-Based Alias Analysis](#). ICST, **Distinguished Paper Award**, 2021

*Efficient dynamic slicing for mobile**



1. Light instrumentation → only basic blocks, not every statement, no fields
2. Post-execution analysis of the trace
 - Recovers dataflows
 - Static, but analyzes one execution path only

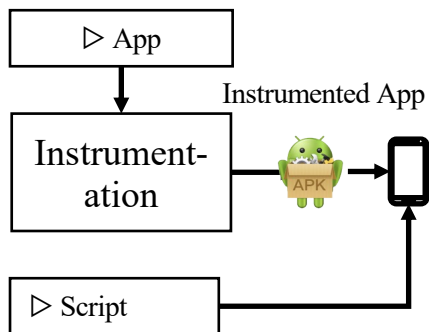


* Khaled Ahmed, Mieszko Lis, and Julia Rubin. [MANDOLINE: Dynamic Slicing of Android Applications with Trace-Based Alias Analysis](#). ICST, **Distinguished Paper Award**, 2021

*Efficient dynamic slicing for mobile**



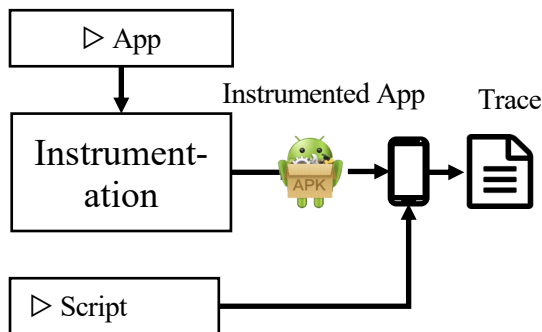
1. Light instrumentation → only basic blocks, not every statement, no fields
2. Post-execution analysis of the trace
 - Recovers dataflows
 - Static, but analyzes one execution path only



*Efficient dynamic slicing for mobile**



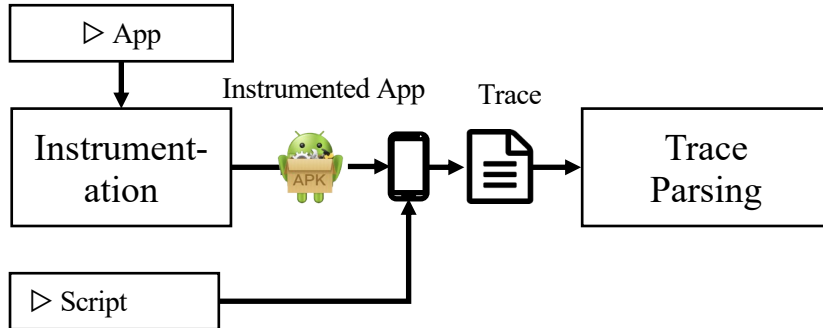
1. Light instrumentation → only basic blocks, not every statement, no fields
2. Post-execution analysis of the trace
 - Recovers dataflows
 - Static, but analyzes one execution path only



*Efficient dynamic slicing for mobile**



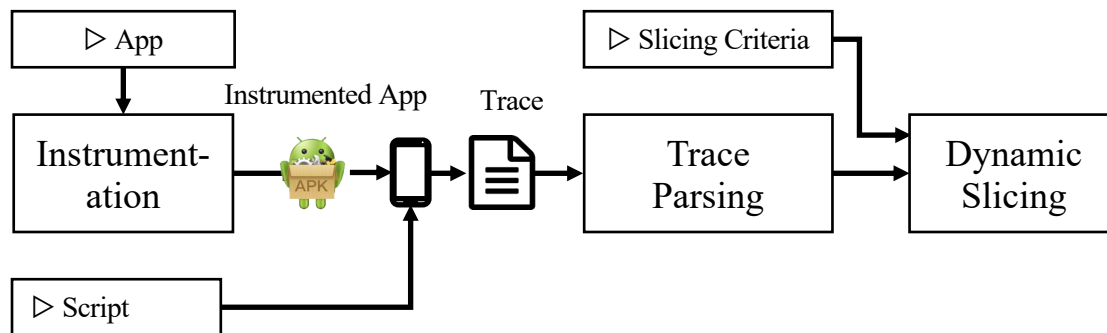
1. Light instrumentation → only basic blocks, not every statement, no fields
2. Post-execution analysis of the trace
 - Recovers dataflows
 - Static, but analyzes one execution path only



Efficient dynamic slicing for mobile*



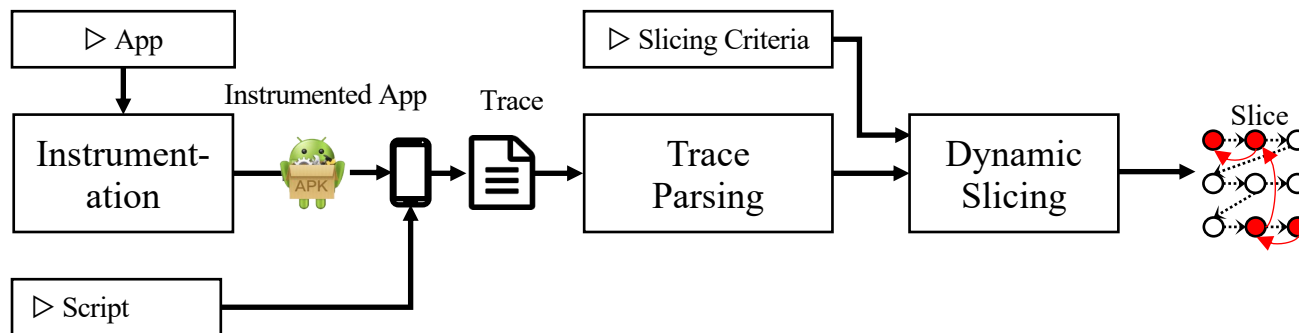
1. Light instrumentation → only basic blocks, not every statement, no fields
2. Post-execution analysis of the trace
 - Recovers dataflows
 - Static, but analyzes one execution path only



Efficient dynamic slicing for mobile*



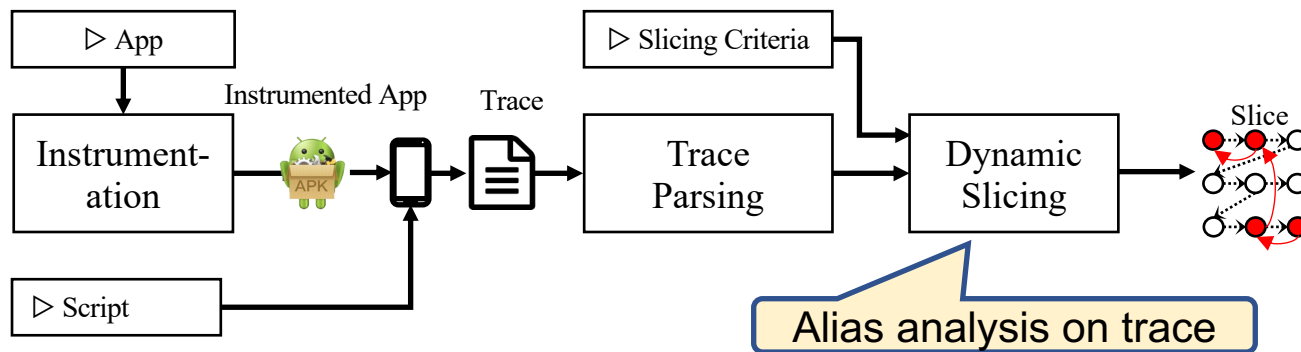
1. Light instrumentation → only basic blocks, not every statement, no fields
2. Post-execution analysis of the trace
 - Recovers dataflows
 - Static, but analyzes one execution path only



Efficient dynamic slicing for mobile*



1. Light instrumentation → only basic blocks, not every statement, no fields
2. Post-execution analysis of the trace
 - Recovers dataflows
 - Static, but analyzes one execution path only



Trace based alias analysis

1 `jane.age = 15`

2 `john = jane`

3 `john.age = 25`

4 `res = jane.age`

`john aliases jane`
`john.age aliases jane.age`



Trace based alias analysis



```
1  jane.age = 15
2  john      = jane
3  john.age  = 25
4  res = jane.age
```



Trace based alias analysis



```
1  jane.age = 15
2  john      = jane
3  john.age  = 25
4  res = jane.age      use
```

Trace based alias analysis

1 jane.age = 15

2 john = jane

3 john.age = 25

4 res = jane.age

Search for john.age too



use

Trace based alias analysis

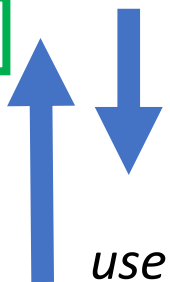
1 jane.age = 15

2 john = jane

3 john.age = 25

4 res = jane.age

Search for john.age too



definition



use

Trace based alias analysis

1 jane.age = 15

2 john = jane

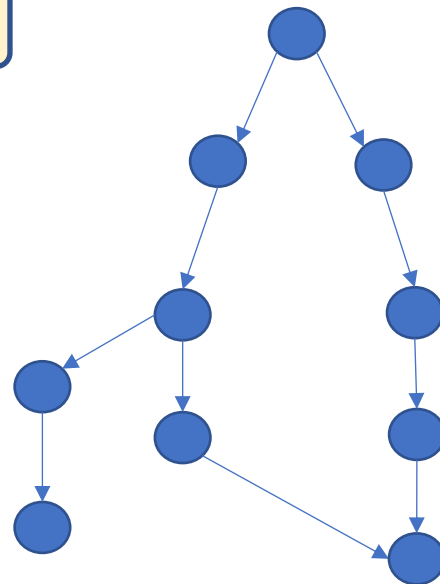
3 john.age = 25

4 res = jane.age

Search for john.age too

definition ✓

use



Trace based alias analysis

1 jane.age = 15

2 john = jane

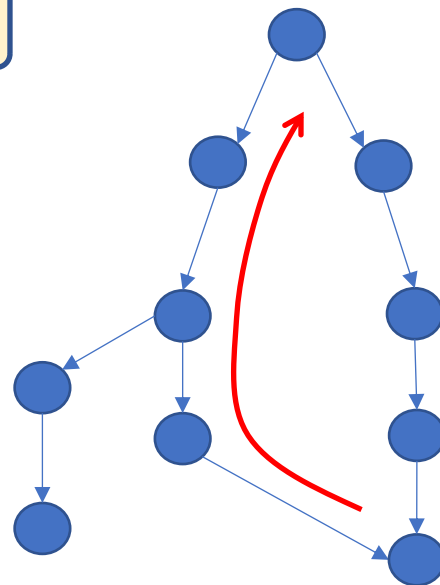
3 john.age = 25

4 res = jane.age

Search for john.age too

definition ✓

use



Trace based alias analysis

1 jane.age = 15

2 john = jane

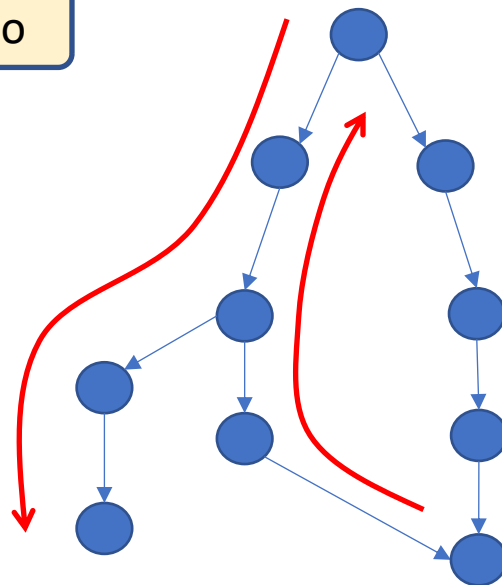
3 john.age = 25

4 res = jane.age

Search for john.age too

definition ✓

use



Trace based alias analysis

1 jane.age = 15

2 john = jane

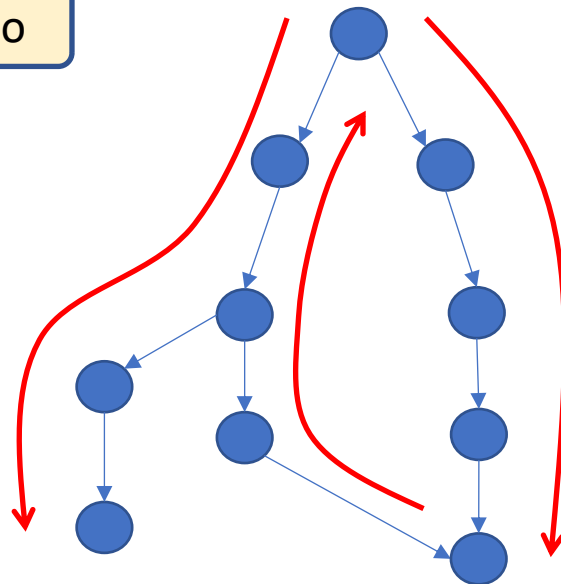
3 john.age = 25

4 res = jane.age

Search for john.age too

definition ✓

use



Trace based alias analysis

1 jane.age = 15

2 john = jane

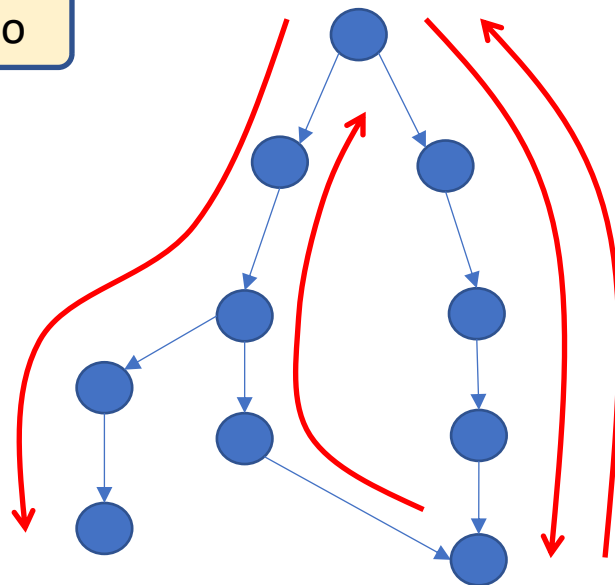
3 john.age = 25

4 res = jane.age

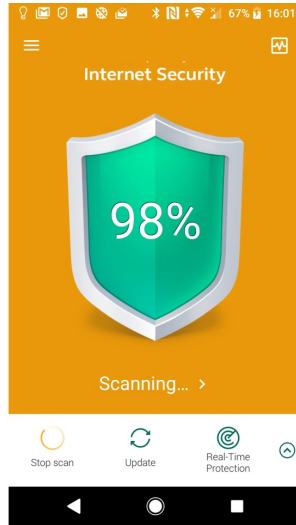
Search for john.age too

definition ✓

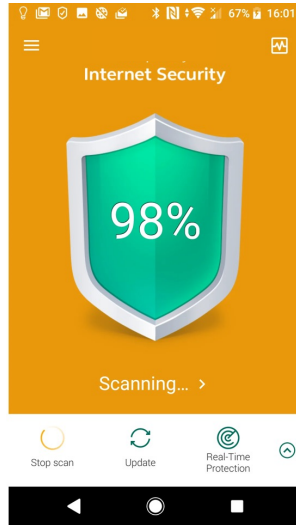
use



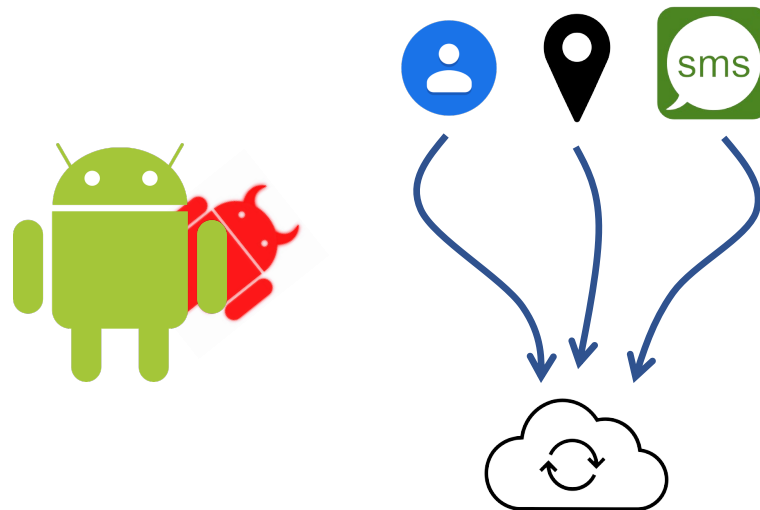
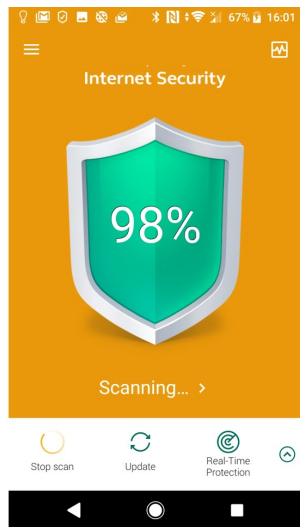
Application in malware analysis



Application in malware analysis



Application in malware analysis



Slice separates **malware** from **benign**

MANDOLINE: Dynamic slicing for Android

- Paper: [Mandoline: Dynamic Slicing of Android Applications with Trace-Based Alias Analysis](#). ICST, Distinguished Paper Award, 2021
- Code: <https://github.com/resess/Mandoline>



Slicer4J: Dynamic slicing for Java

- Paper: [Slicer4J: A Dynamic Slicer for Java](#). ESEC/FSE, tools track, 2021
- Code: <https://github.com/resess/Slicer4J>



IntelliJ plugin for Slicer4J: Capstone project

- Integration of slicing, slice visualization, and slice navigation in the most popular IDE!

What's next

Automated malware summarization / detection

- Using slices to summarize the behavior of malware / detect new samples

Slicing for software maintainance

- Slicing to compare versions of code

Interested? Join us!

Contact : Khaled Ahmed, <https://khaled-e-a.github.io>
khaleddea@ece.ubc.ca

Julia Rubin, <https://people.ece.ubc.ca/mjulia/>,
mjulia@ece.ubc.ca