Die nächste Generation von Schwachstellenscannern: präzise, einfach, anpassbar

Prof. Dr. Eric Bodden

@profbodden

PADERBORN UNIVERSITY
The University for the Information Society

Fraunhofer IEM

HEINZ NIXDORF INSTITUT
UNIVERSITÄT PADERBORN

SECURE SOFTWARE ENGINEERING GROUP

HEINZ NIXDORF INSTITUT
UNIVERSITÄT PADERBORN

SECURE
SOFTWARE ENGINEERING
GROUP

Lisa Nguyen

Fraunhofer IEM

Stefan Krüger

UNIVERSITÄT PADERBORN

Karim Ali

UNIVERSITY OF ALBERTA

Ben Livshits

Imperial College London

Justin Smith

NC STATE UNIVERSITY

Emerson Murphy-Hill

NC STATE UNIVERSITY

Mira Mezini

TECHNISCHE UNIVERSITÄT DARMSTADT

Agenda

- Traditionelle Statische Analyse
- Demand-driven Static Analysis
- Just-in-time Analysis für IDE-Integration
- Beispiel: Crypto-Checker “CogniCrypt”

2015 DHS Study:

“90% of security incidents result from exploits against defects in software”

- Nicht Krypto
- Nicht Netzwerke
- Nicht Hardware

Annahmen für diesen Vortrag

- Wir befassen uns mit Statischer Analyse
  - Keine Dynamische Analyse, kein Fuzzing, etc.

- Wir analysieren “gutartigen Code”, suchen nach Schwachstellen
  - keine böswillige Obfuskierung

```
                                                                    char
                                                                _3141592654[3141
        ],__3141[3141];_314159[31415],_3141[31415];main(){register char*
      _3_141,*_3_1415, *_3__1415; register int _314,_31415,__31415,*_31,
    _3_14159,__3_1415;*_3141592654=__31415=2,_3141592654[0][_3141592654
   -1]=1[__3141]=5;__3_1415=1;do{_3_14159=_314=0,__31415++;for( _31415
  =0;_31415<(3,14-4)*__31415;_31415++)_31415[_3141]=_314159[_31415]= -
1;_3141[*_314159=_3_14159]=_314;_3_141=_3141592654+__3_1415;_3_1415=
__3_1415    +__3141;for                (_31415 = 3141-
          __3_1415  ;                  _31415;_31415--
          ,_3_141 ++,                  _3_1415++){_314
          +=_314<<2 ;                  _314<<=1;_314+=
         *_3_1415;_31                   =_314159+_314;
         if(!(*_31+1)                   )* _31 =_314 /
         __31415,_314                   [_3141]=_314 %
         __31415 ;* (                   _3__1415=_3_141
        )+= *_3_1415                     = *_31;while(*
        _3__1415 >=                      31415/3141 ) *
        _3__1415+= -                     10,(*--_3__1415
         )++;_314=_314                  [_3141]; if ( !
         _3_14159 && *                  _3_1415)_3_14159
         =1,__3_1415 =                  3141-_31415;}if(
         _314+(__31415                   >>1)>=__31415 )
         while ( ++ *                    _3_141==3141/314
        )*_3_141--=0                     ;}while(_3_14159
        ) ; { char *                     __3_14= "3.1415";
       write((3,1),                      (--*__3_14,__3_14
       ),(_3_14159                        ++,++_3_14159))+
      3.1415926; }                        for ( _31415 = 1;
     _31415<3141-                         1;_31415++)write(
    31415% 314-(                          3,14),_3141592654[
  _31415    ] +                          "0123456789","314"
  [ 3]+1)-_314;                          puts((*_3141592654=0
,_3141592654))                           ;_314= *"3.141592";}
```

http://www.ioccc.org/

Application Security Testing Tools

Figure 1. Magic Quadrant for Application Security Testing

Source: Gartner (February 2017)

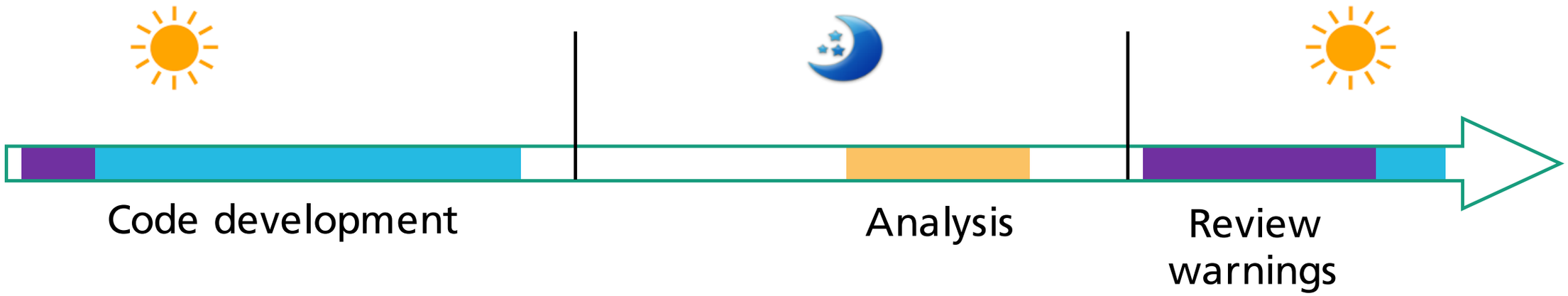
Static Application Security Testing (SAST)

Vorteile von SAST (theoretisch)

- Funktioniert bei unvollständigem Code
- Gibt Entwicklern direktes Feedback
- Kann das gesamte Programm abdecken
- Erkennt viele Arten von Schwachstellen
- Kann sinnvolle Fixes vorschlagen

Static Analysis in der Praxis

Stand der Technik oftmals: Batch-Analyse



SCA Analysis Results - With TeamMentor

Filter Set: Security Auditor View My Issues

4286 | 2335 | 8 | 1334 | 7963

Critical (4286)

Group By: Category

- Command Injection - [0 / 3]
  - Exec.java:107 (Command Injection)
  - Exec.java:289 (Command Injection)
  - WSDLScanning.java:148 (Command Injection)
- Cross-Site Scripting: DOM - [0 / 2]
- Cross-Site Scripting: Persistent - [0 / 3227]
- Cross-Site Scripting: Reflected - [0 / 822]
- Password Management: Hardcoded Password - [0 / 19
- Path Manipulation - [0 / 6]
- Privacy Violation - [0 / 148]
- Race Condition: Singleton Member Field - [0 / 8]
- SQL Injection - [0 / 49]
- XPath Injection - [0 / 2]

Advanced...

Analysis Evidence

- ParameterParser.java:615 - getParameterValues(return)
- ParameterParser.java:615 - Assignment to values
- ParameterParser.java:623 - Return values[0]
- ParameterParser.java:597 - getRawParameter(return)
- ParameterParser.java:597 - Return
- Challenge2Screen.java:641 - getRawParameter(return)
- Challenge2Screen.java:641 - Assignment to protocol

Project Summary | Exec.java

```
        ByteArrayOutputStream output = new ByteArrayOutputStream();
        ByteArrayOutputStream errors = new ByteArrayOutputStream();
        ExecResults results = new ExecResults(Arrays.asList(command).toStr
        BitSet interrupted = new BitSet(1);
        boolean lazyQuit = false;
        ThreadWatcher watcher;

        try
        {
            // start the command
            child = Runtime.getRuntime().exec(command);

            // get the streams in and out of the command
            InputStream processIn = child.getInputStream();
            InputStream processError = child.getErrorStream();
            OutputStream processOut = child.getOutputStream();

            // start the clock running
            if (timeout > 0)
            {
                watcher = new ThreadWatcher(child, interrupted, timeout);
                new Thread(watcher).start();
            }

            // Write to the child process' input stream
            if ((input != null) && !input.equals(""))
            {
                try
                {
                    processOut.write(input.getBytes());
                    processOut.flush();
```

Issue | Issue | Recom | History | Diagra | Correla | Screen

Issue: Exec.java:107 (Command Injection)

User:

Analysis:

Command Injection (Input Validation and

The method execOptions() in Exec.java calls exec() with a command built from untrusted data. This call can cause the

SCA Analysis Results – With TeamMentor

Filter Set: Security Auditor View My Issues

4286 | 2335 | 8 | 1334 | 7963

Critical (4286)

Group By: Category

- Command Injection – [0 / 3]
  - Exec.java:107 (Command Injection)
  - Exec.java:289 (Command Injection)
  - WSDLScanning.java:148 (Command Injection)
- Cross-Site Scripting: DOM – [0 / 2]
- Cross-Site Scripting: Persistent – [0 / 3227]
- Cross-Site Scripting: Reflected – [0 / 822]
- Password Management: Hardcoded Password – [0 / 19
- Path Manipulation – [0 / 6]
- Privacy Violation – [0 / 148]
- Race Condition: Singleton Member Field – [0 / 8]
- SQL Injection – [0 / 49]
- XPath Injection – [0 / 2]

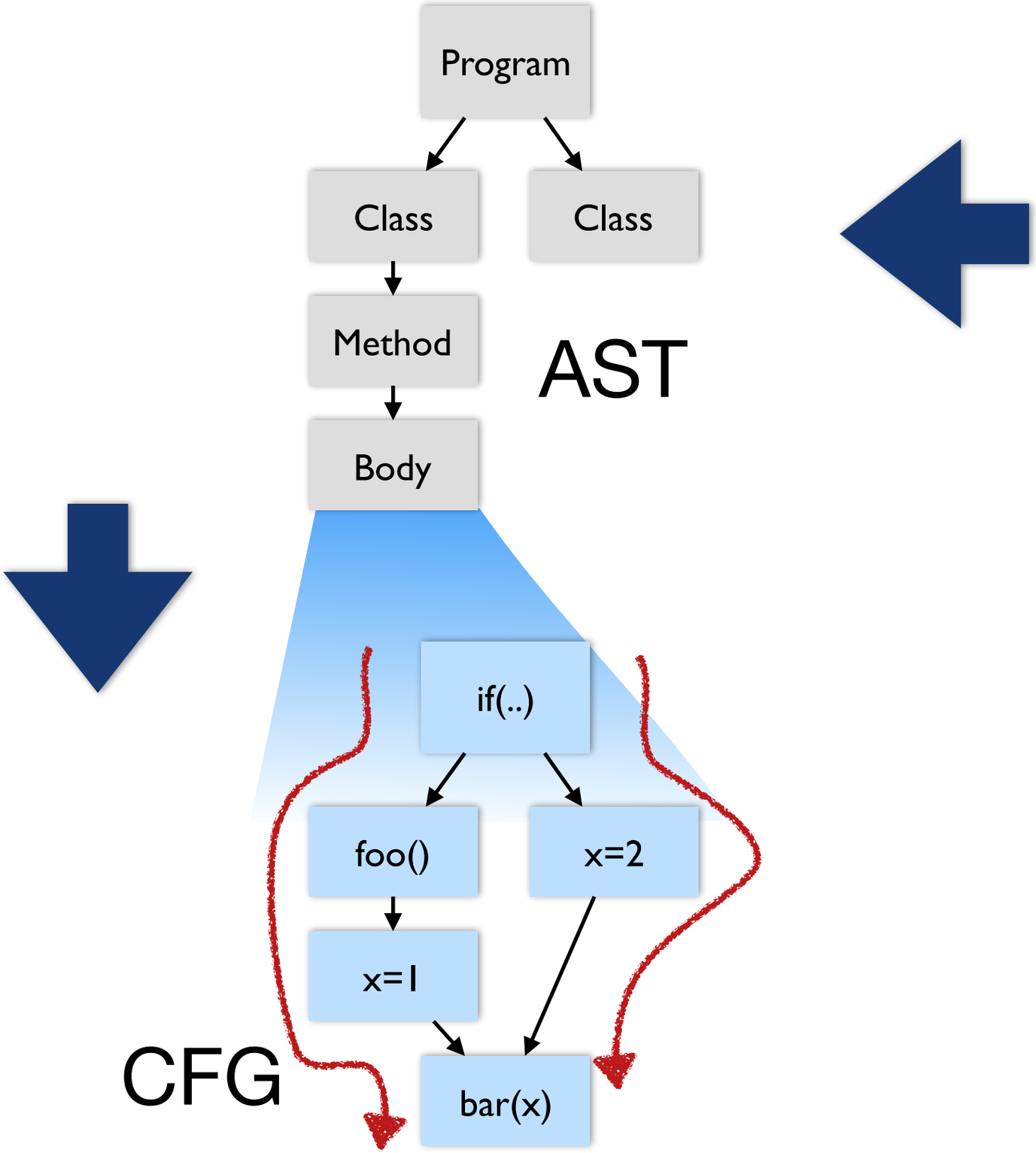
Project Summary

```
            ByteArrayOu
            ByteArrayOu
            ExecResults
            BitSet inte
            boolean lazy
            ThreadWatche

            try
            {
                // star
                child =

                // get
                InputSt
                InputSt
                OutputS

                // star
                if (time
                {
                    wate
                    new
                }
```

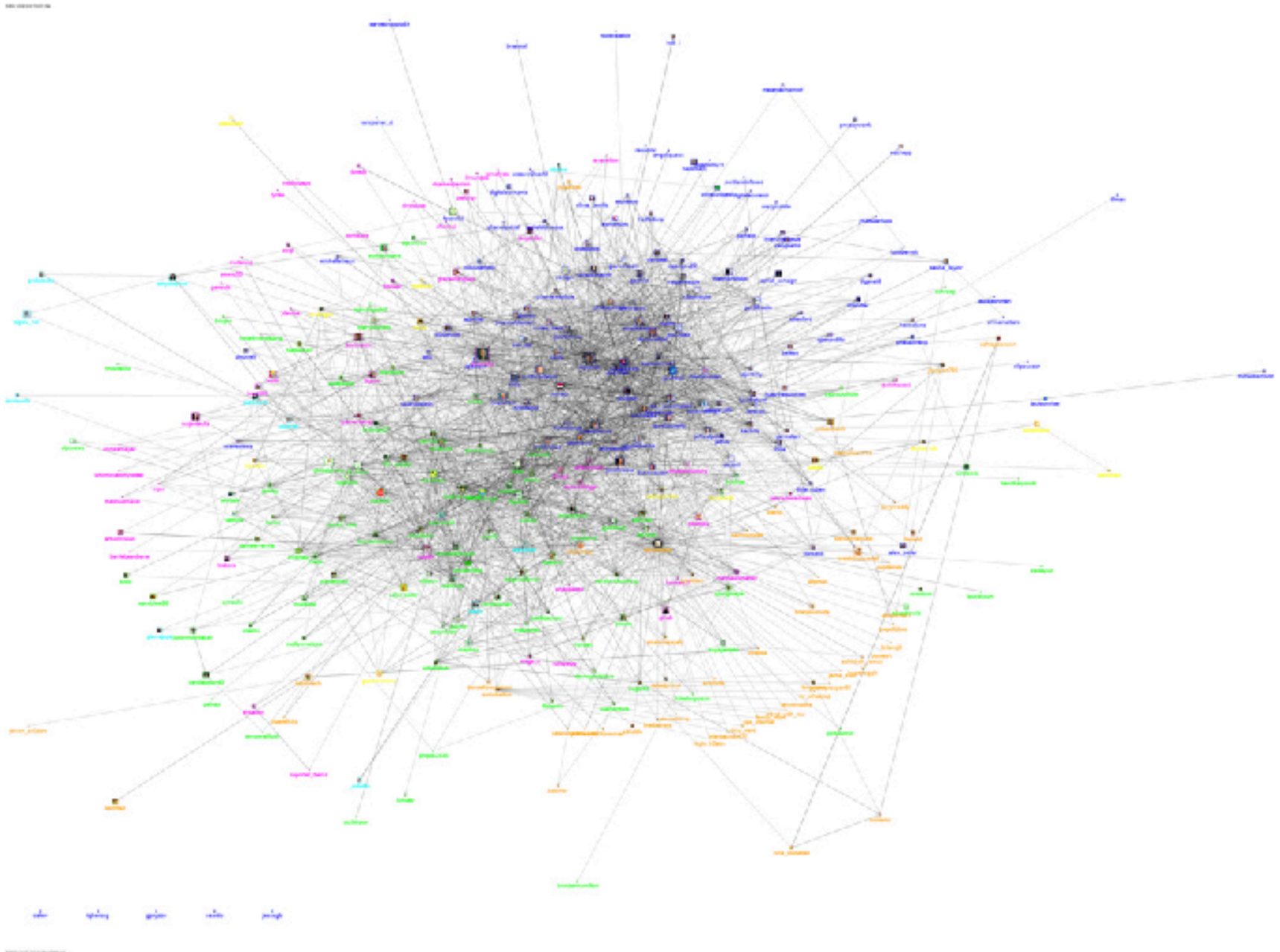
Warum benötigen viele Analysetools soviel Zeit?

HEINZ NIXDORF INSTITUT
UNIVERSITÄT PADERBORN

SECURE
SOFTWARE ENGINEERING
GROUP

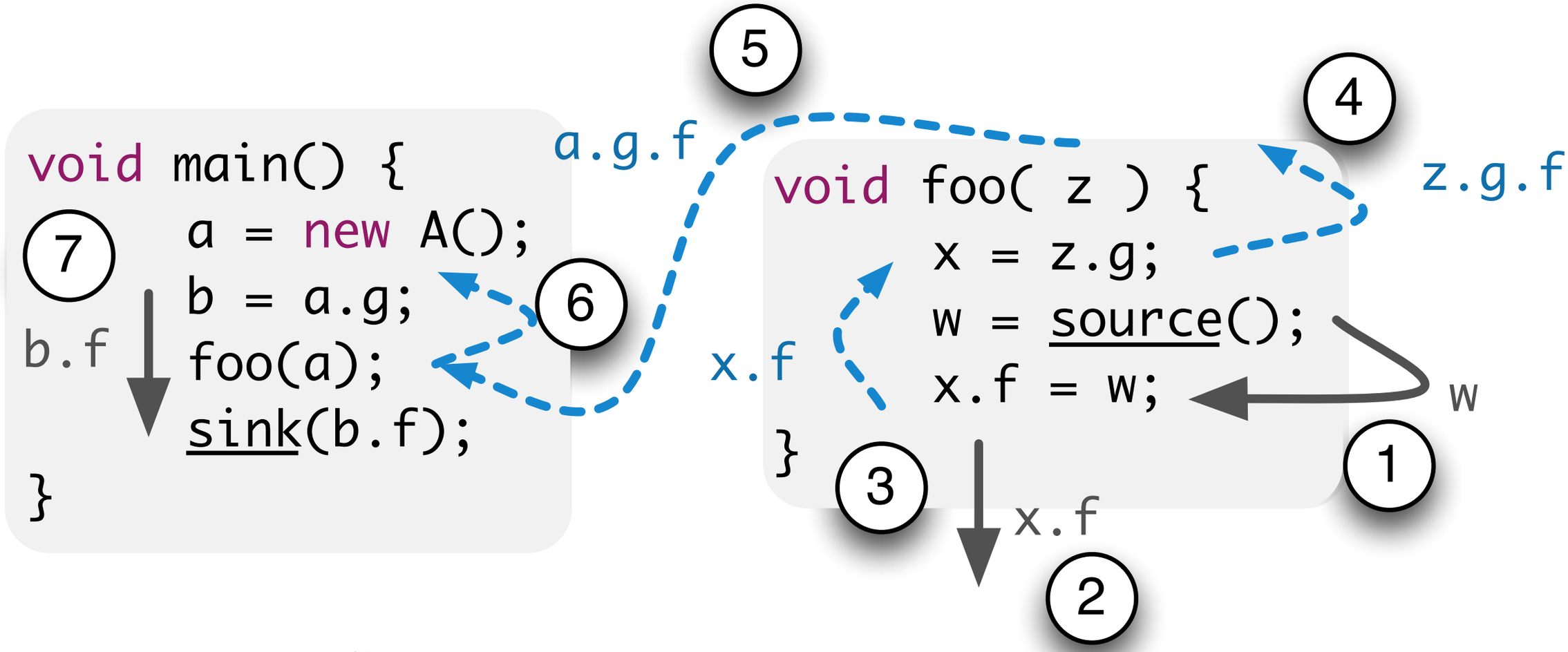

Program
Class
Class
Method
Body
AST
if(..)
foo()
x=2
x=1
bar(x)
CFG

Reale Programmgraphen sing riesig Millionen von Knoten und Kanten!

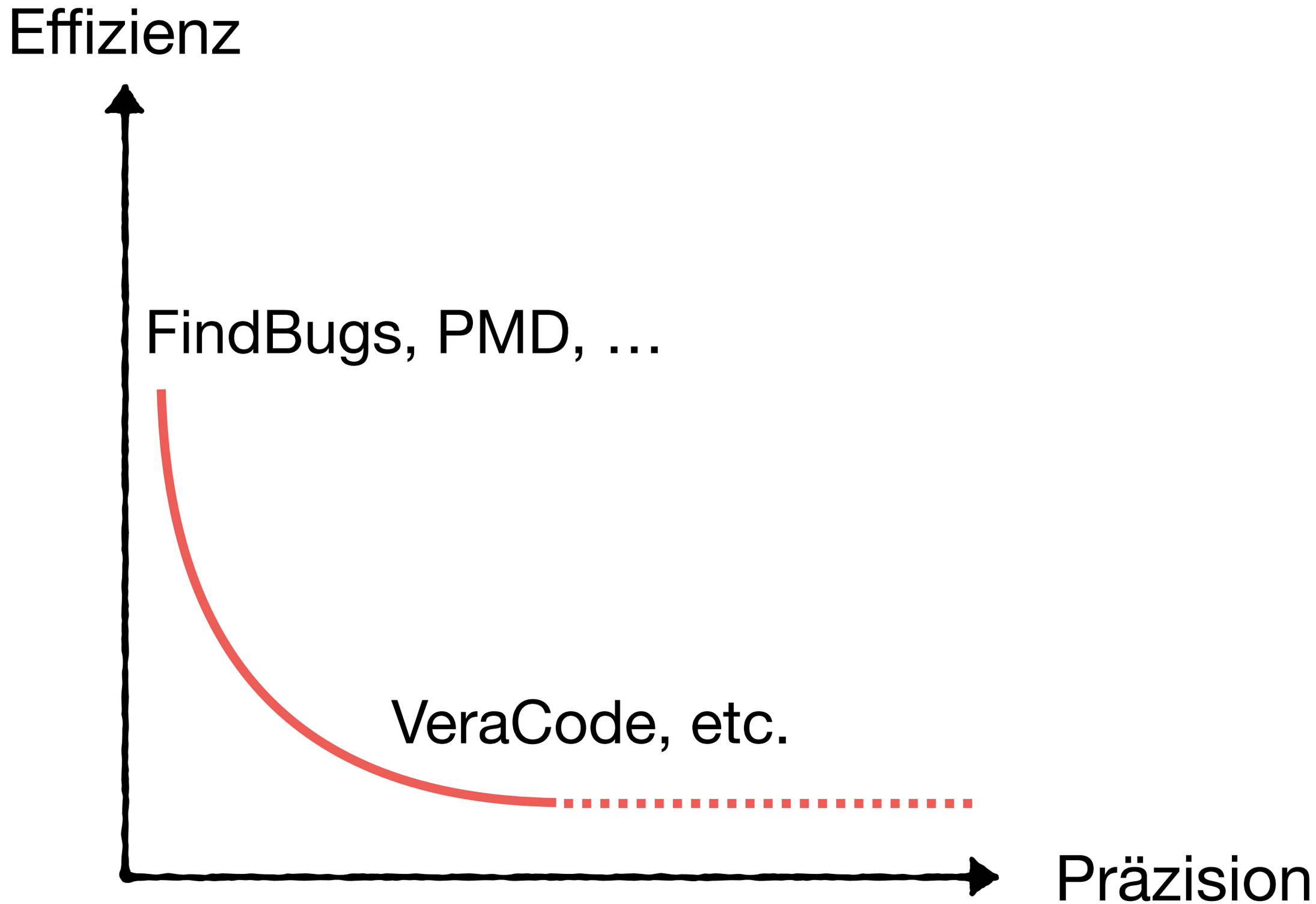


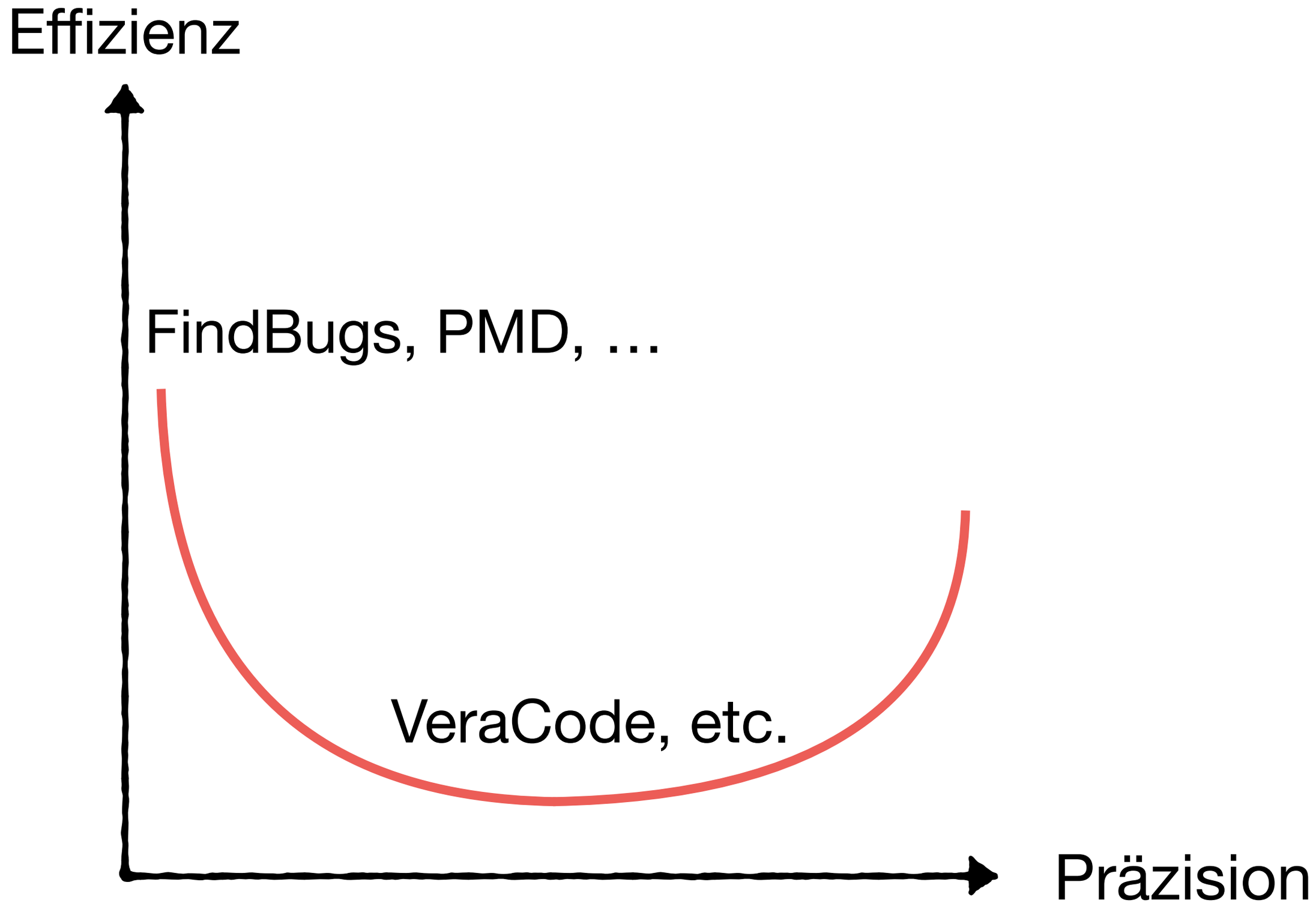
Auch: “Aliasing”-Problem

```
String secret = mySecret();

String public = secret;

print(public);
```



Leckt hier was?


Effizienz
FindBugs, PMD, …
VeraCode, etc.
Präzision


Effizienz
FindBugs, PMD, …
VeraCode, etc.
Präzision

Beispiel:
Feld-Sensitive Analyse

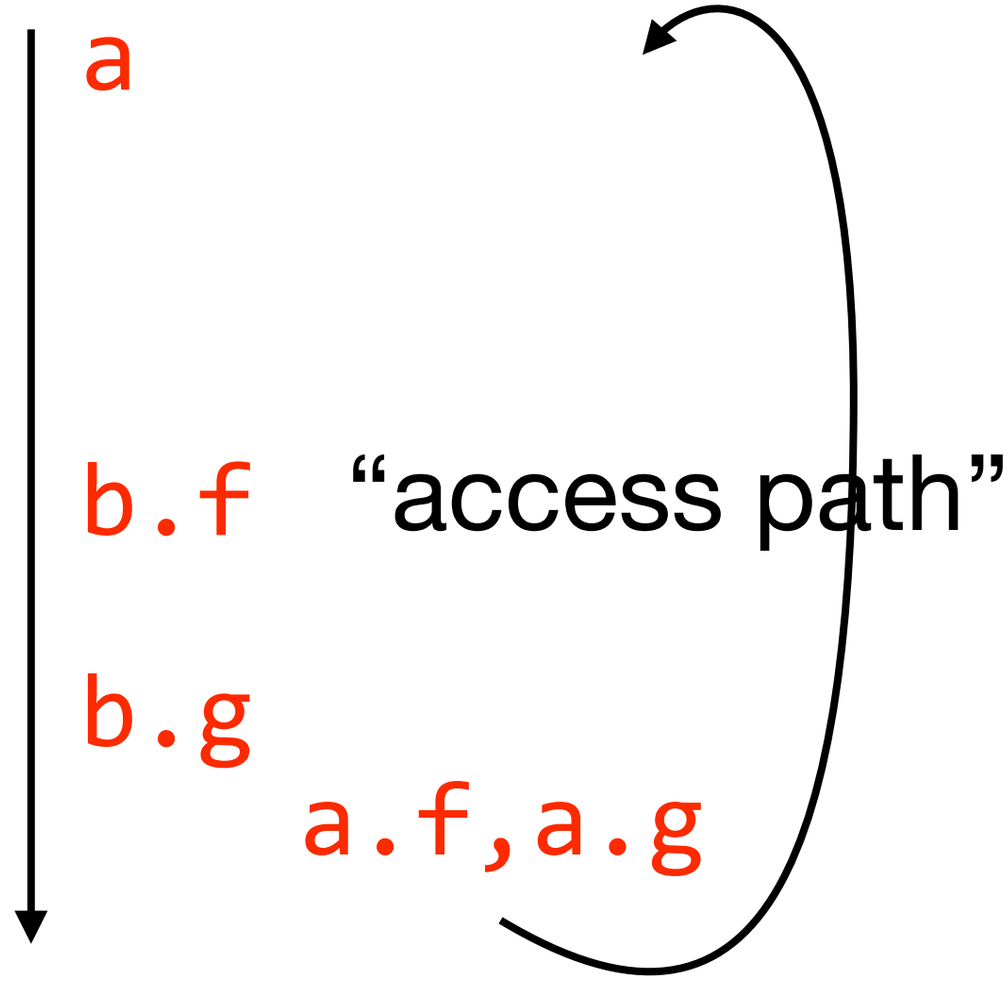
```
a = password();
while(..) {
  A b = new A();
  if(..)
    b.f = a;
  else
    b.g = a;
  a = b;
  b = null;
}
```

a

b.f

b.g

a.f,a.g

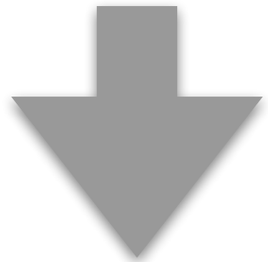


```
a = password();
while(..) {
   A b = new A();
   if(..)
     b.f = a;
   else
     b.g = a;
   a = b;
   b = null;
}
```

```
a.f

b.f.f, b.g.f

b.f.g, b.g.g

a.f.g,a.f.f,
b.g.f,b.g.g

...
```

Jahrzehntelange Behelfslösung: k-limiting

`b.f.f, b.f.g`



`b.f.*`

0 l = k

k-limiting ist aber keine Lösung

```
o.f = secret();

print(o.g);
```

{o.*}

Access-Path Abstraction: Scaling Field-Sensitive Data-Flow Analysis With Unbounded Access Paths

Johannes Lerch*, Johannes Späth†, Eric Bodden*† and Mira Mezini*‡
*Technische Universität Darmstadt, †Fraunhofer SIT, ‡Lancaster University
*†Darmstadt, Germany, ‡Lancaster, United Kingdom
*{lastname}@cs.tu-darmstadt.de, †{firstname.lastname}@sit.fraunhofer.de

***Abstract*—Precise data-flow analyses frequently model field accesses through access paths with varying length. While using longer access paths increases precision, their size must be bounded to assure termination, and should anyway be small to enable a scalable analysis.**

We present Access-Path Abstraction, which for the first time combines efficiency with maximal precision. At control-flow merge points Access-Path Abstraction represents all those access paths that are rooted at the same base variable through this base variable only. The full access paths are reconstructed on demand where required. This makes it unnecessary to bound access paths to a fixed maximal length.

Experiments with Stanford SecuriBench and the Java Class Library compare our open-source implementation against a field-based approach and against a field-sensitive approach that uses bounded access paths. The results show that the proposed approach scales as well as a field-based approach, whereas the approach using bounded access paths runs out of memory.

I. Introduction

Static program analyses and especially data-flow analyses usually have to consider values being assigned to and read from local variables and fields. While local variables are often simple to track, the systematic handling of fields can be complex. Techniques for modeling fields can be distinguished into field-based and field-sensitive approaches. Field-based techniques model a field access $a.f$ simply by the field's name f, plus potentially its declaring type—a course grain approach that ignores the base object a's identity. Field-sensitive techniques, on the other hand, include the base variable a in the static abstraction, potentially increasing analysis precision as fields belonging to different base objects can be distinguished.

While many existing analyses restrict their field-sensitivity to a single level, more precise analyses represent static information through entire *access paths* – a base variable followed by a finite sequence of field accesses [1]–[3]. For instance, assume a taint analysis determining whether (and where) the example shown in Figure 1a might print the password. Here the analysis must distinguish access paths of length $k \geq 2$ (k is the maximal length of the finite sequence) to determine that a leak can occur only at the first print statement. While the use of access paths can increase precision, one must generally bound k to a finite maximum, as otherwise loops or recursive data structures such as linked lists might cause abstractions such as `l.next.prev.next...` to grow indefinitely, which would cause the analysis not to terminate.

A common approach to deal with infinite chains of field accesses is k-limiting [4], which includes in the abstraction only the first k nested field accesses, abstracting from all others. If fields are read, the analyses frequently assume that any field accessible beyond the first k fields may relate to the tracked information. Hence, k-limiting introduces an over-approximation that becomes less precise for smaller values of k. In contrast, a high k value means the analysis will distinguish more states. In this work, we present experiments, which show that analyses can run out of gigabytes of main memory when analyzing real-world programs even with small k values.

To address the problems raised above, we present Access-Path Abstraction, a novel and generic approach for handling field-sensitive analysis abstractions *without the need for k-limiting*. To keep access paths finite and small, at control-flow merge points Access-Path Abstraction represents all those access paths that are rooted at the same base variable through this base variable only. In a summary-based inter-procedural analysis this leads to fewer and yet highly reusable summaries, which can speed up the analysis significantly: A procedure summary for a base object a can represent information for all access paths rooted in a. To maintain the precision of field-sensitive analyses, Access-Path Abstraction reconstructs the full access paths on demand where required.

We present Access-Path Abstraction as a novel extension to the IFDS framework for inter-procedural finite distributive subset problems [5] and hereafter use IFDS-APA as an acronym for this extension. We provide an open-source implementation on top of the IFDS/IDE solver Heros [6].[1]

In experiments using Stanford SecuriBench and the Java Class Library as benchmarks, we compare IFDS-APA against a field-based approach and against a field-sensitive approach using k-limiting, both in terms of scalability and analysis time. The results show that IFDS-APA scales as well as a field-based approach, whereas a field-sensitive approach using k-limiting runs out of memory.

To recap, this work presents the following original contributions:

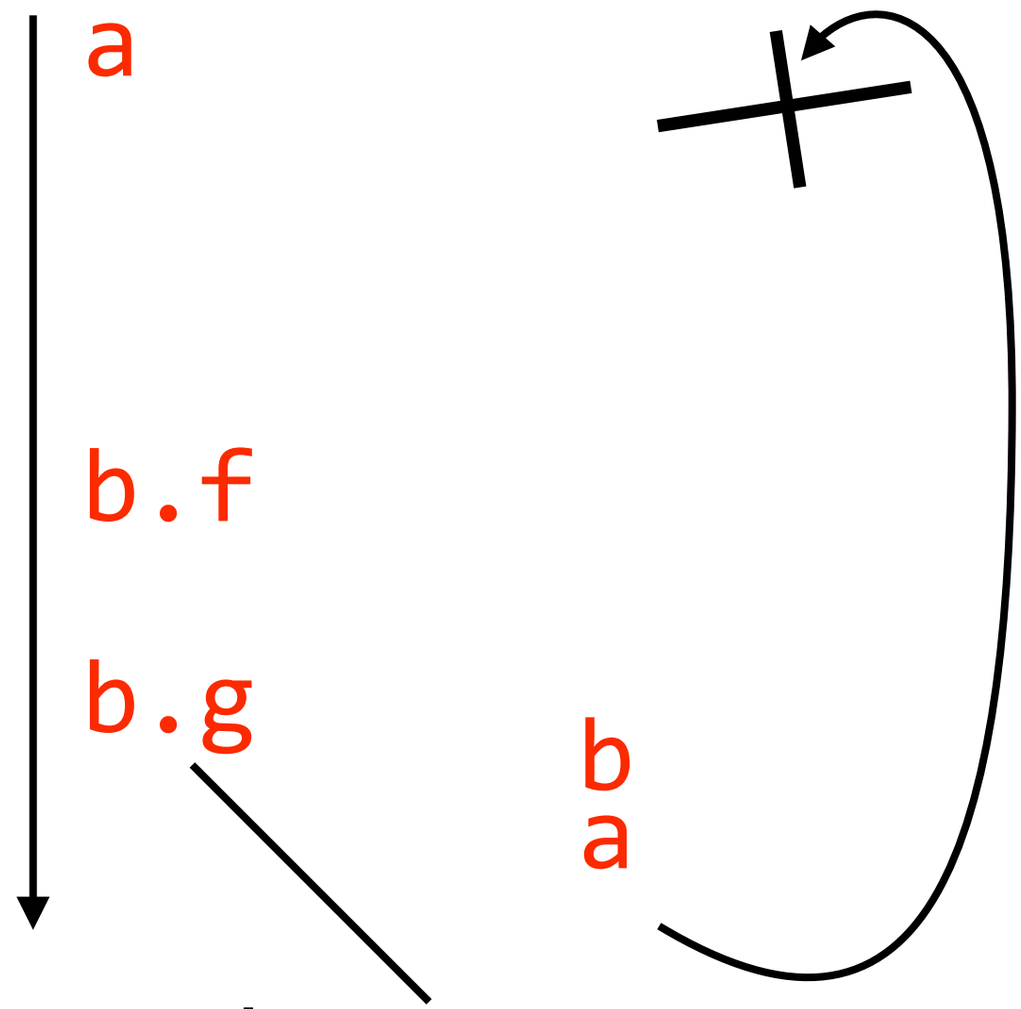
- a novel extension to the IFDS framework enabling scalable and precise field-sensitive analysis without the need for k-limiting,
- a full open-source implementation, and
- extensive experiments comparing the performance of the proposed approach to a field-based and a field-sensitive approach with k-limiting.

[1]Heros is hosted on GitHub: https://github.com/secure-software-engineering/heros Our implementation is available there.

Unsere Lösung: Access Path Abstraction [ASE 2015]

- Access-paths “on demand” nur dort berechnet wo benötigt
- Effektiv ein auto-tuning des Parameters *k*

```
a = password();
while(..) {
   A b = new A();
   if(..)
     b.f = a;
   else
      b.g = a;
    a = b;
      b = null;
}
```



Trick vielfältig anwendbar:

Berechne jede Information *on demand* nur wenn wirklich benötigt

On-demand Analyse

- Berechne **Access Paths**
  nur wo nötig

[ASE 2015]

- Berechne **Pointer-Information**
  nur wo nötig

[ECOOP 2016]

- Berechne **String-Werte**
  nur wo nötig

[TR 2017]

- Berechne **Call Graph-Kanten**
  nur wo nötig

in Arbeit…

können einander aufrufen!

Beispiel:

SQL-Injection

HEINZ NIXDORF INSTITUT
UNIVERSITÄT PADERBORN

SECURE
SOFTWARE ENGINEERING
GROUP

```
Scanner scanner = new Scanner(System.in);
User user = new User();

System.out.print("Enter username: ");
String user.name = scanner.nextLine();

System.out.print("Enter password: ");
String user.password = scanner.nextLine();

//Try to authenticate
if(!auth(user, conn)) return;
```

```
private static boolean auth(User u, Connection conn) {

        String query = "SELECT * FROM Users WHERE Username='" + u.name
                     + "' AND Password='" + u.password + "‘‘";
        return conn.createStatement().executeQuery(query).next();

}
```

query

```
Scanner scanner = new Scanner(System.in);
User user = new User();

System.out.print("Enter username: ");
String user.name = scanner.nextLine();

System.out.print("Enter password: ");
String user.password = scanner.nextLine();

//Try to authenticate
if(!auth(user, conn)) return;
```

user.name

user.password

u.name

u.password

```
private static boolean auth(User u, Connection conn) {

      String query = "SELECT * FROM Users WHERE Username='" + u.name
                   + "' AND Password='" + u.password + "‘‘";
      return conn.createStatement().executeQuery(query).next();

}
```

```
Scanner scanner = new Scanner(System.in);
User user = new User();

System.out.print("Enter username: ");
String user.name = scanner.nextLine();

System.out.print("Enter password: ");
String user.password = scanner.nextLine();

//Try to authenticate
if(!auth(user, conn)) return;
```

user.password

user.name

```
private static boolean auth(User u, Connection conn) {

        String query = "SELECT * FROM Users WHERE Username='" + u.name
                     + "' AND Password='" + u.password + "‘";
        return conn.createStatement().executeQuery(query).next();

}
```

```java
Scanner scanner = new Scanner(System.in);
User user = new User();

System.out.print("Enter username: ");
String user.name = scanner.nextLine();

System.out.print("Enter password: ");
String user.password = scanner.nextLine();

//Try to authenticate
if(!auth(user, conn)) return;
```

```java
private static boolean auth(User u, Connection conn) {

                        String query = "SELECT * FROM Users WHERE Username='" + u.name
                                     + "' AND Password='" + u.password + "'";
                          return conn.createStatement().executeQuery(query).next();

}
```

→ Dieselbe Laufzeit!

Just-In-Time Static Analysis

[ISSTA 2017]

HEINZ NIXDORF INSTITUT
UNIVERSITÄT PADERBORN

SECURE SOFTWARE ENGINEERING GROUP

JIT Analysis Framework

revolutioniert die Art und Weise,
wie Entwickler mit SAST-Tools arbeiten

Java - DemoProject/src/de/mo/MainActivity.java - Eclipse Platform

File Edit Refactor Source Navigate Search Project Run Window Help

Quick Access | Resource | Java

Packa... | Type ...

- DemoProject
  - Android 4.2.2
  - Android Private Libraries
  - src
    - de.mo
      - MainActivity.java
      - OtherActivity.java
  - gen [Generated Java Files]
  - Referenced Libraries
  - bin
  - lib
  - libs
  - res
  - AndroidManifest.xml
  - ic_launcher-web.png
  - proguard-project.txt
  - project.properties

MainActivity.java | OtherActivity.java

```
import android.app.Activity;

public class MainActivity extends Activity {

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_button2);

        TelephonyManager mgr = (TelephonyManager) this.getSystemService(TELEPHONY_SERVICE);
        String secret = mgr.getDeviceId();
        OtherActivity.secret = secret;

        final SmsManager sm = SmsManager.getDefault();
        String send = "IMEI: " + secret;
        sm.sendTextMessage("+49 1234", null, send, null, null);
        doSomething(secret);
    }

    private void doSomething(String s) {
        Log.d("LEAK", s);
    }

    @Override
    protected void onPause() {
        TelephonyManager mgr = (TelephonyManager) this.getSystemService(TELEPHONY_SERVICE);
        String imei = mgr.getDeviceId();
        Log.d("LEAK", imei);
    }
}
```

Detail

Statement	Location

Console | Overview

Id	Source	Sink	Source location	Sink location
1	String secret = mgr.getDeviceId();	sm.sendTextMessage("+49 1234", null, send, null, null);	18:MainActivity.java	23:MainActivity.java
2	String imei = mgr.getDeviceId();	Log.d("LEAK", imei);	34:MainActivity.java	35:MainActivity.java
3	String secret = mgr.getDeviceId();	Log.d("LEAK", s);	18:MainActivity.java	28:MainActivity.java
4	String secret = mgr.getDeviceId();	sm.sendTextMessage("+49 1234", null, secret, null, null);	18:MainActivity.java	17:OtherActivity.java

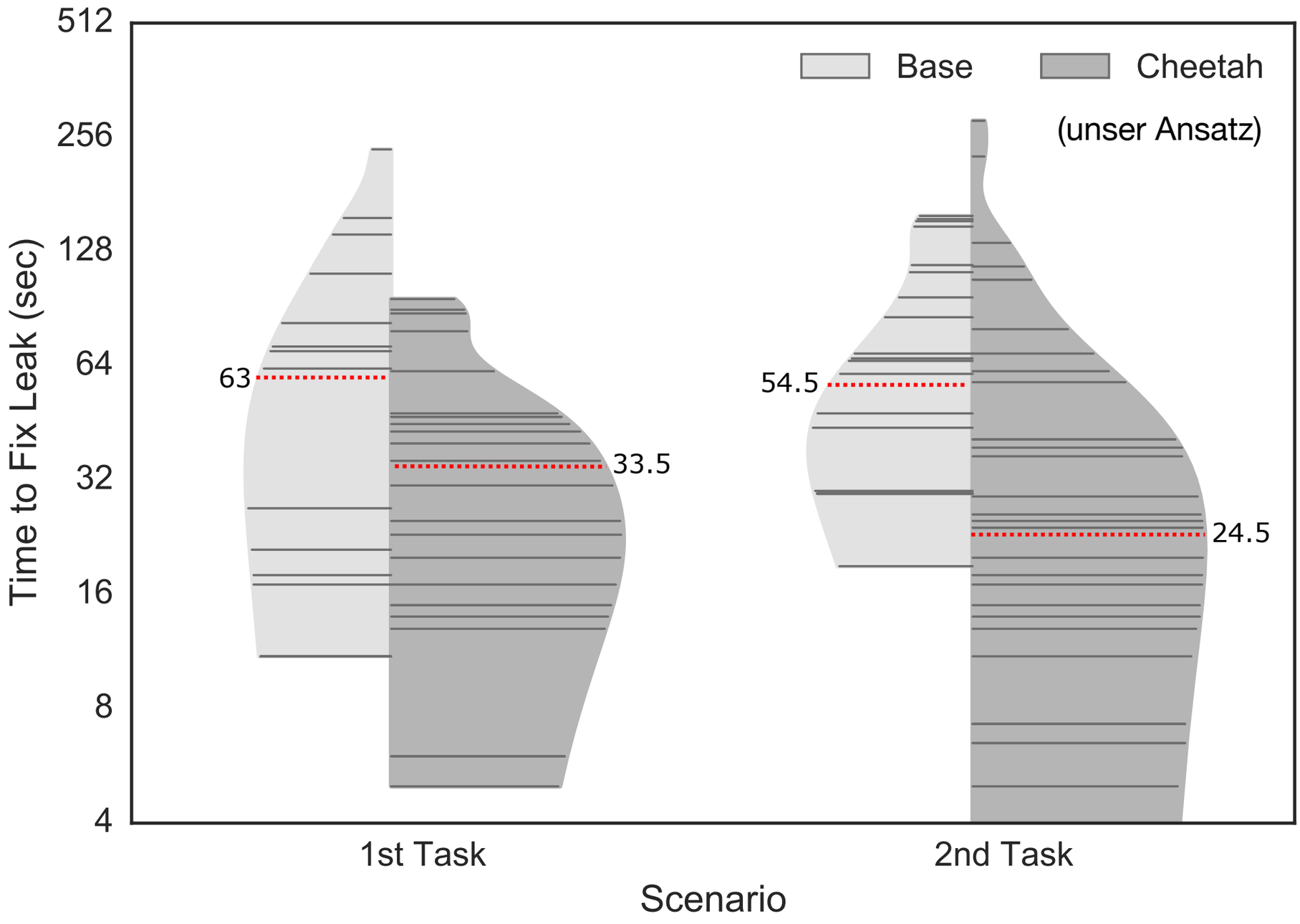
de.mo.MainActivity.java - DemoProject/src

https://www.youtube.com/watch?v=AMq9sFo7gjc

HEINZ NIXDORF INSTITUT
UNIVERSITÄT PADERBORN

SECURE SOFTWARE ENGINEERING GROUP

Mit der JIT-Analyse beheben Entwickler Bugs schneller!



Ziel für die Zukunft:

Assistenzsystem,
das den Entwickler
gezielt trainiert

CROSSING

Domänen-spezifischer Prototyp:

www.eclipse.org/cognicrypt/

HEINZ NIXDORF INSTITUT
UNIVERSITÄT PADERBORN

SECURE
SOFTWARE ENGINEERING
GROUP

How to securely connect to a server?

Encryption mode?

User accounts

Encryption vs Hashing?

Payment info.

Application

Sensitive user documents

How to encrypt data?

Salted hashing?

Entwickler in der freien Laufbahn

83% of 269 Vulnerabilities are due to misuse of crypto libraries
[Lazar et al., APSys ’14]

Even Amazon & Paypal misuse SSL certificate validation
[Georgiev et al., CCS ‘12]

88% of ~12,000 Android apps misuse crypto APIs
[Egele et al., CCS ‘13]

Designziele:

- Effizient und präzise Fehlbenutzungen von Crypto-APIs feststellen
- Einfach auf neue Crypto-APIs anpassbar

CogniCrypt

ORACLE

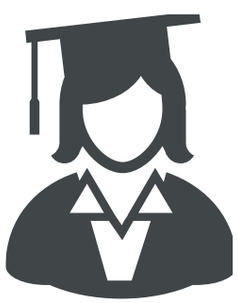
Neue Domänen-spezifische Definitionssprache *CrySL*

```
SPEC javax.crypto.KeyGenerator

OBJECTS
  java.lang.String algorithm;
  int keySize;
  javax.crypto.SecretKey key;

EVENTS
  g1: getInstance(algorithm);
  g2: getInstance(algorithm, _);
  GetInstance := g1 | g2;

  i1: init(keySize);
  i2: init(keySize, _);
  i3: init(_);
  i4: init(_, _);
  Init := i1 | i2 | i3 | i4;

  GenKey: key = generateKey();
```

Eine spec für KeyGenerator

… spricht über diese Werte

… und diese Events

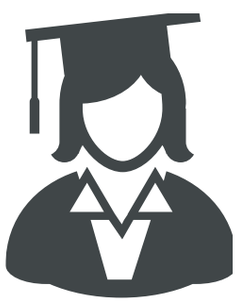


Event-Reihenfolge

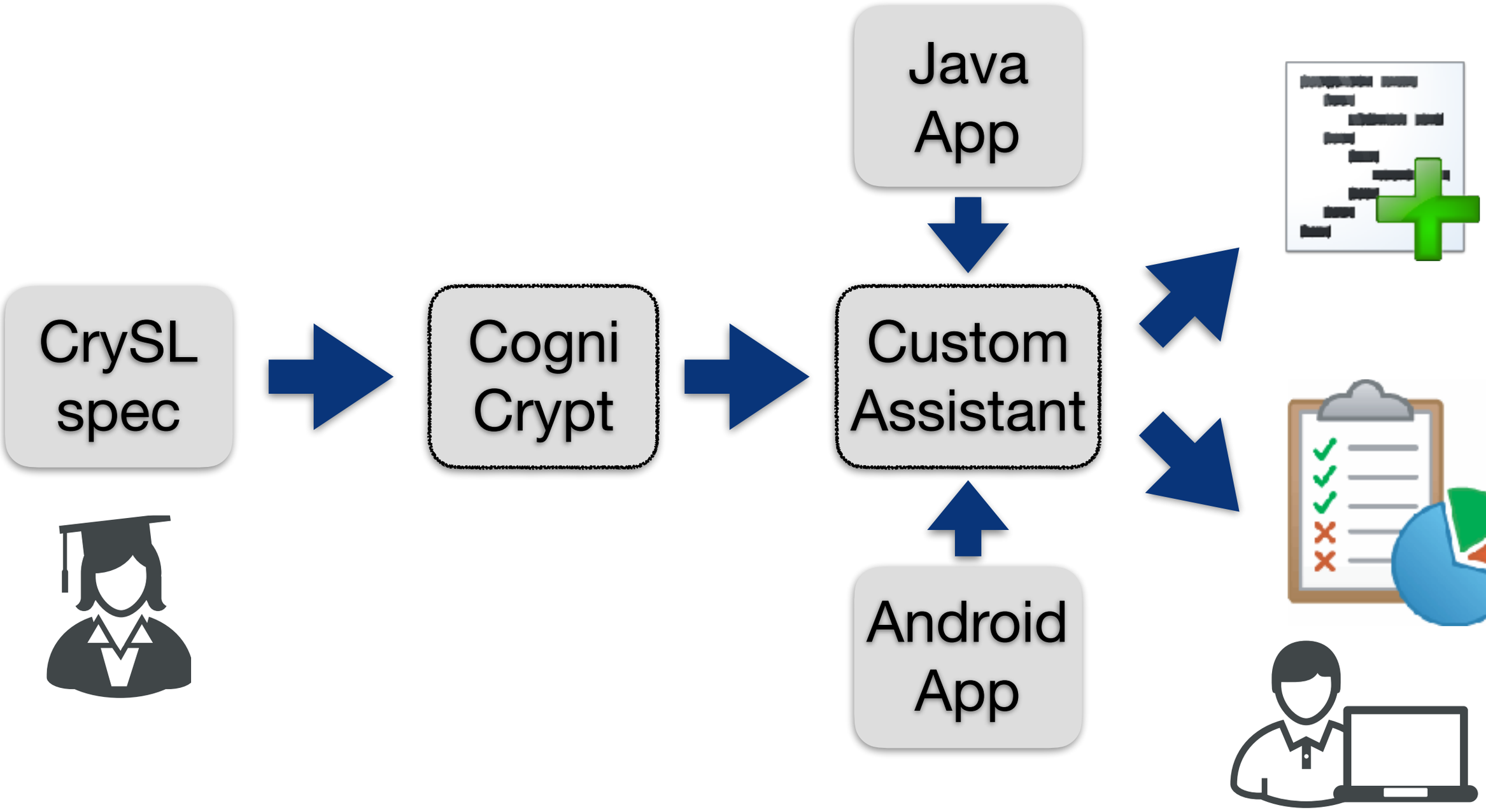
Parameter Constraints

```
ORDER
  GetInstance, Init?, GenKey

CONSTRAINTS
  algorithm in {"AES", "Blowfish"};
  algorithm in {"AES"} => keySize in {128, 192, 256};
  algorithm in {"Blowfish"} => keySize in {128, 192,
      256, 320, 384, 448};
```

Aufruf:
Suchen Mitstreiter, die
Java Crypto APIs gut beherrschen!



CogniCrypt Workflow



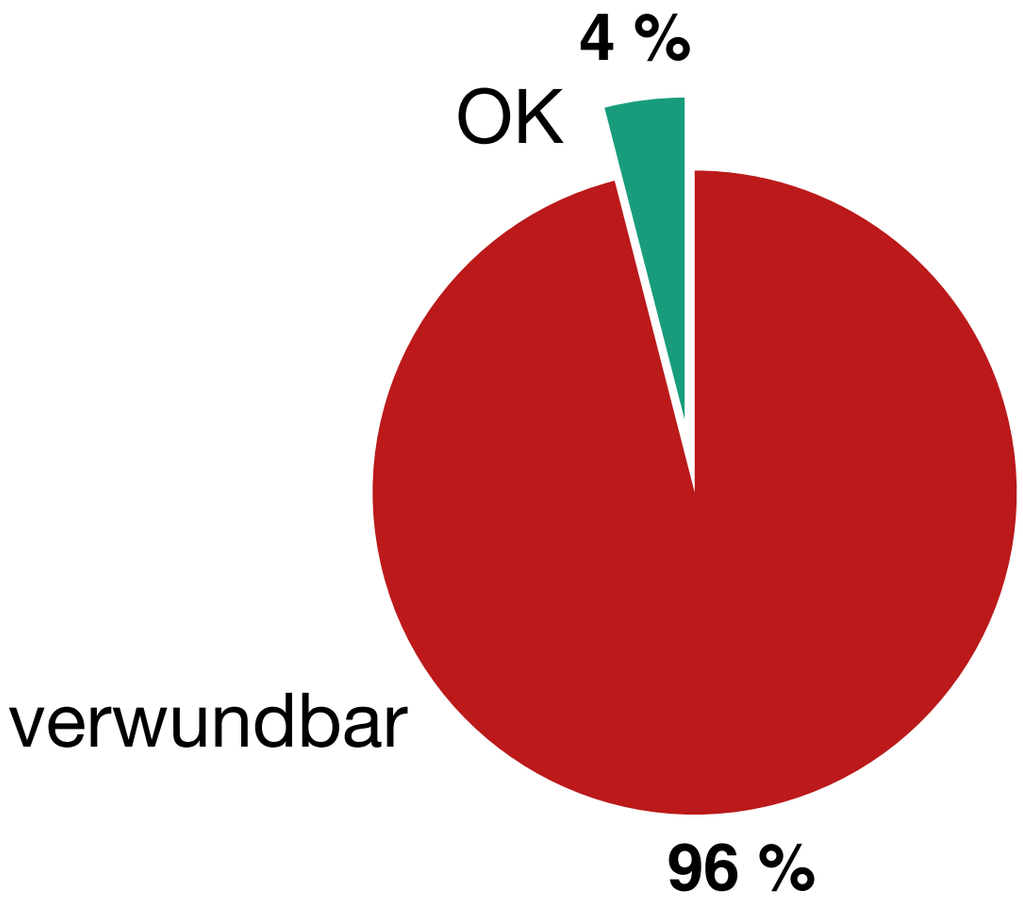
Demand-driven analysis durch die Bank:

- Access Paths / Feldzugriffe
- Alias-Information
- String-Wert-Berechnung

In Planung:
On-Demand Call Graph-Konstruktion

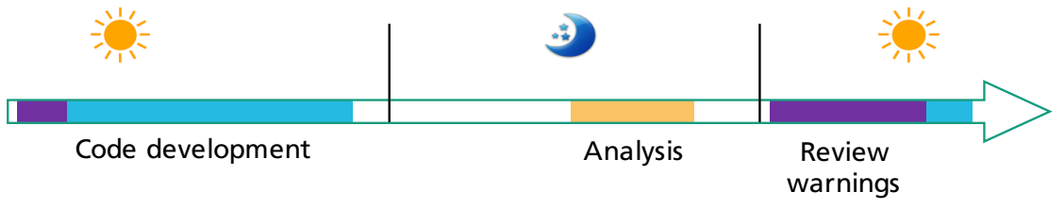
Evaluation

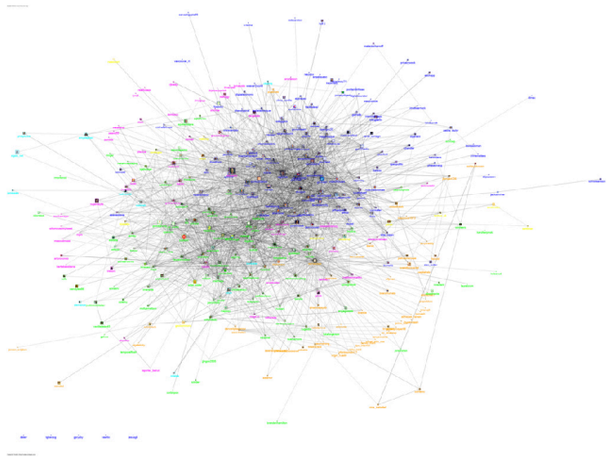
CogniCrypt auf 10,000 Android Apps



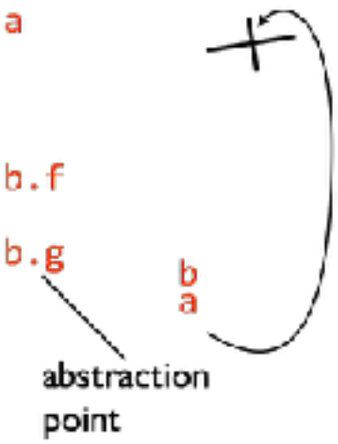
Häufigstes Problem:
Nutzung von MD5/
SHA1, AES/ECB, …

Analysezeit im Schnitt
ca. 100s





```
a = password();
while(..) {
  A b = new A();
  if(..)
    b.f = a;
  else
    b.g = a;
  a = b;
  b = null;
}
```




```
Scanner scanner = new Scanner(System.in);
User user = new User();

System.out.print("Enter username: ");
String user.name = scanner.nextLine();

System.out.print("Enter password: ");
String user.password = scanner.nextLine();

//Try to authenticate
if(!auth(user, conn)) return;
```

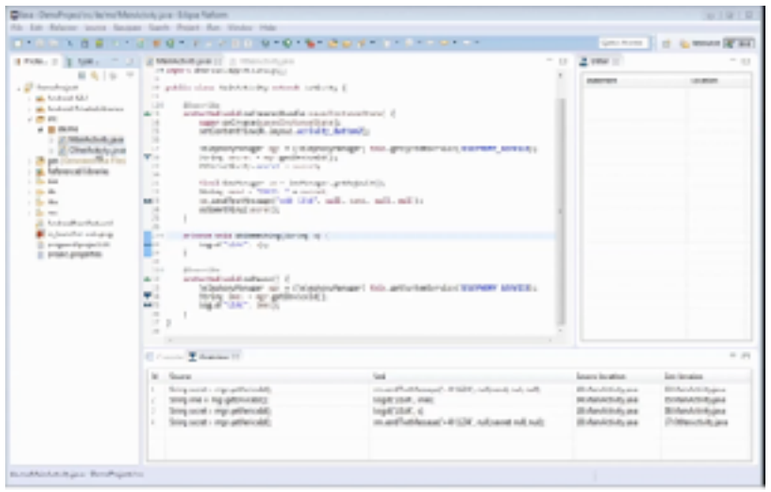
user.name

user.password

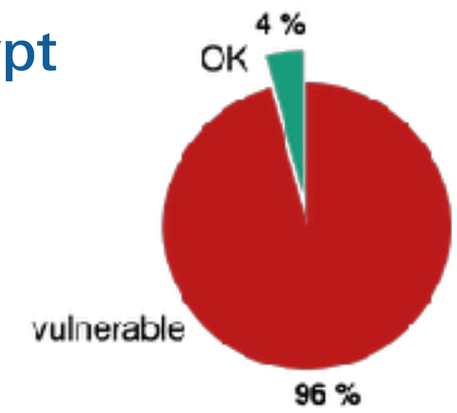
u.password

u.name

```
private static boolean auth(User u, Connection conn) {

        String query = "SELECT * FROM Users WHERE Username='" + u.name
                     + "' AND Password='" + u.password + "'";
        return conn.createStatement().executeQuery(query).next();

}
```



CogniCrypt



<100s

Prof. Dr. Eric Bodden
Chair for Software Engineering
Heinz Nixdorf Institut
Zukunftsmeile 1
33102 Paderborn

Telefon: +49 5251 60-3313
eric.bodden@uni-paderborn.de

@profbodden

https://www.hni.uni-paderborn.de/swt/
https://blogs.uni-paderborn.de/sse/