SECURE YOUR CODE AS FAST AS YOU RELEASE

How ShiftLeft is able to analyze a million lines of code in just under 15 minutes

Dresden, November 11, 2019

Dr.-Ing. Max Leuthäuser
max@shiftleft.io

ShiftLeft

About me

2019 - Senior Software Developer, ShiftLeft GmbH (ShiftLeft.io)

2018 - Consultant, Software Developer, itemis
Language Engineer (Xtext, Xtend, EMF/Ecore, Eclipse; mainly Automotive)

2017 - Dr.-Ing., TU Dresden

Outline

1. ShiftLeft in a nutshell
2. Getting technical
3. Other things I want to talk about

Application is THE Attack Surface

ars TECHNICA

Muni system hacker hit others by scanning for year-old Java vulnerability

Bloomberg Technology

Uber Hack Shows Vulnerability of Software Code-Sharing Services

TC

Mixpanel analytics accidentally slurped up passwords

WIRED

LILY HAY NEWMAN SECURITY 09.14.17 01:27 PM

EQUIFAX OFFICIALLY HAS NO EXCUSE

ZDNet

The attack happened at around the same time as one security researcher, known as Revolver, disclosed a local file inclusion flaw on the AdultFriendFinder site.

benefitspro

Accidental data breaches remain the leading cause of loss

InfoWorld FROM IDG

How you could be leaking your secrets onto GitHub

WIRED

ROBERT MCMILLAN BUSINESS 04.11.14 06:30 AM

HOW HEARTBLEED BROKE THE INTERNET — AND WHY IT CAN HAPPEN AGAIN

GIZMODO

Wag Left User Data Exposed

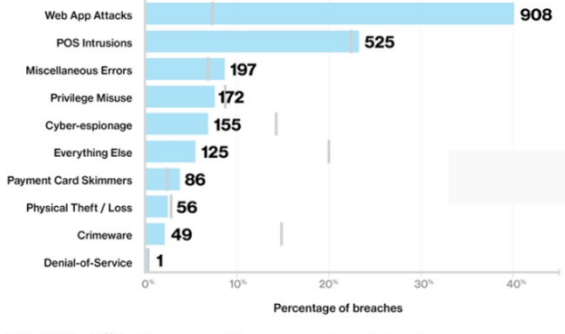
"A vast majority of the attacks will be on the custom code in an application"

Gartner

verizon digital media services

Verizon DBIR 2016: Web Application Attacks are the #1 Source of Data Breaches

Percentage and count of attacks that resulted in data breaches per pattern, DBIR 2016

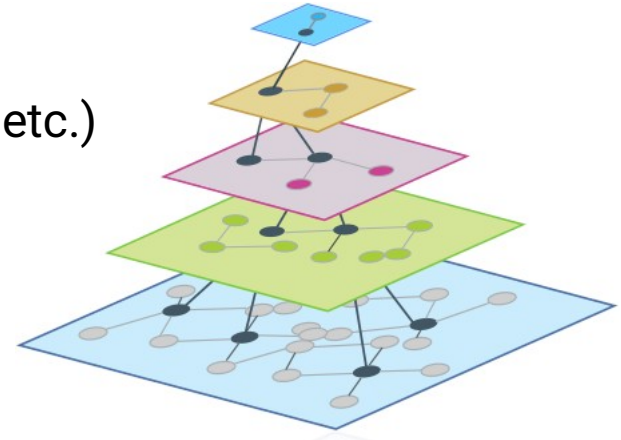


ShiftLeft in a Nutshell

1

Code analysis solution that finds:

- Business logic flaws
  (Auth bypasses, Insecure Direct Object References, etc.)
- Insider threats, rootkits & back-doors
- Data flows & critical leakages
- Vulnerabilities in your code & all its dependencies

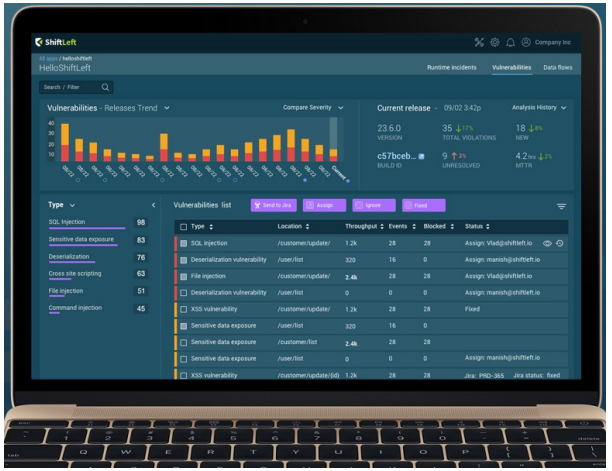


ShiftLeft in a Nutshell

1

Automation

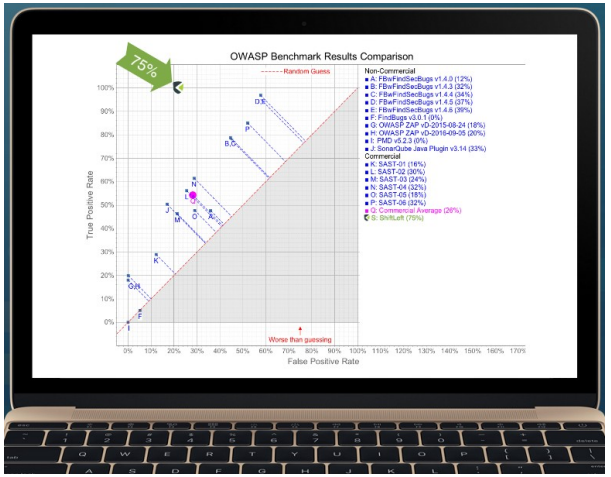
- Code analysis at the speed of CI/CD
- Too late once your stuff is deployed at the customer side



ShiftLeft in a Nutshell

Results

Achieves highest Static Application Security Testing (SAST) score ever on the OWASP Benchmark



The OWASP Benchmark for Security Automation is a free and open test suite designed to evaluate the speed, coverage, and accuracy of automated software vulnerability detection tools and services.

Four possible test outcomes in the Benchmark:

- Tool correctly identifies a real vulnerability (True Positive)
- Tool fails to identify a real vulnerability (False Negative)
- Tool correctly ignores a false alarm (True Negative)
- Tool fails to ignore a false alarm (False Positive)

ShiftLeft in a Nutshell

1

Developers
- seamlessly insert security into CI/CD (code analysis in minutes, not days)
- fix vulnerabilities faster (get detailed information such as line-of-code for each vulnerability)

AppSec
- protect every version of every release
- increase feature velocity w/o sacrificing security
- identify external data leakages

Code Auditors
- use Turing-complete language to query your application dataflows
- integrate custom security queries into CI/CD
- annotate on your own for customized code analysis

Go Beyond ’`grep`‘ to Analyze Your Code

1

- mine the *Code Property Graph* using a formal graph traversal language
- apply the same query across all your code
  (independent of programming languages)
  `Java, Scala, C, C++, C#, Go, Javascript, Python, LLVM`

Ocular? Joern?

- Goal: provide query language to describe patterns in code
  - to identify bugs and vulnerabilities
  - to help in deeply understanding large programs

- Think of it as an extensible Code Analysis Machine
- Programmable in JVM-based languages (e.g., Java/Scala/Kotlin)
- You can write scripts, language extensions and libraries on top of it

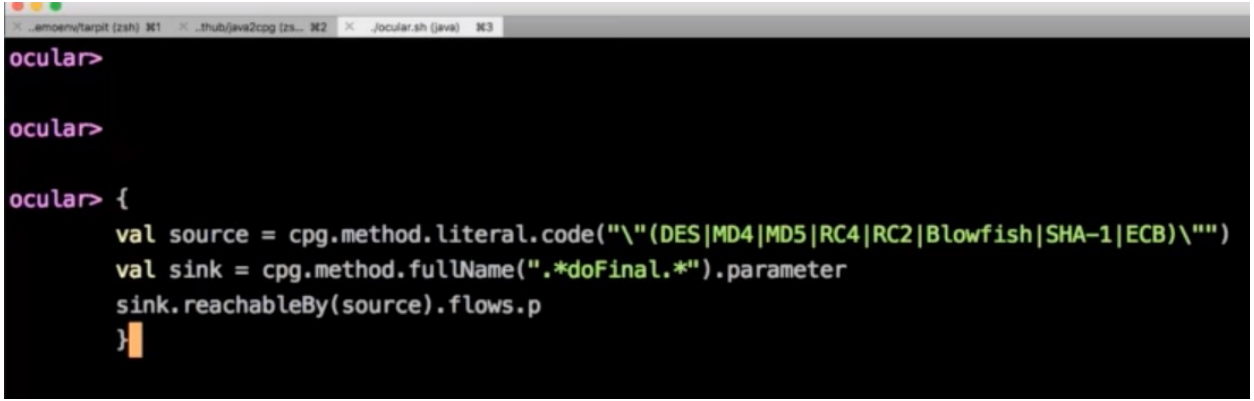
- Joern is Ocular's open-source brother (for C/C++ only)
  See: joern.io/docs

Ocular Example

Weak crypto:

```
/* FLAW: Insecure cryptographic algorithm (DES)
CWE: 327 Use of Broken or Risky Cryptographic Algorithm */
Cipher des = Cipher.getInstance("DES");
SecretKey key = KeyGenerator.getInstance("DES").generateKey();
des.init(Cipher.ENCRYPT_MODE, key);
```

Ocular Example



Ocular Example

```
ocular>

ocular>

ocular> {
        val source = cpg.method.literal.code("\"(DES|MD4|MD5|RC4|RC2|Blowfish|SHA-1|ECB)\"")
        val sink = cpg.method.fullName(".*doFinal.*").parameter
        sink.reachableBy(source).flows.p
        }
2019-03-20 10:43:11.617 [main] INFO mainTasksSize: 10, reachedSource: 1,
source: io.shiftleft.queryprimitives.steps.Steps[Literal, shapeless.HNil] = io.shiftleft.queryprimitiv
sink: io.shiftleft.queryprimitives.steps.types.structure.MethodParameter[shapeless.HNil] = io.shiftlef
res21_2: List[String] = List(
  """
  ____________________________________________________________________________________________
  | tracked            | lineNumber| method               | file                               |
  |============================================================================================|
  | "DES"              | 52        | doGet                | io/shiftleft/tarpit/ServletTarPit.java|
  | param0             | N/A       | getInstance          | javax/crypto/Cipher.java           |
  | ret                | N/A       | getInstance          | javax/crypto/Cipher.java           |
  | getInstance("DES")| 52        | doGet                | io/shiftleft/tarpit/ServletTarPit.java|
  | param1             | N/A       | <operator>.assignment| N/A                                |
  | param0             | N/A       | <operator>.assignment| N/A                                |
  | des                | 52        | doGet                | io/shiftleft/tarpit/ServletTarPit.java|
  | des                | 77        | doGet                | io/shiftleft/tarpit/ServletTarPit.java|
  | this               | N/A       | doFinal              | javax/crypto/Cipher.java           |
"""
)
```

Real-world Example

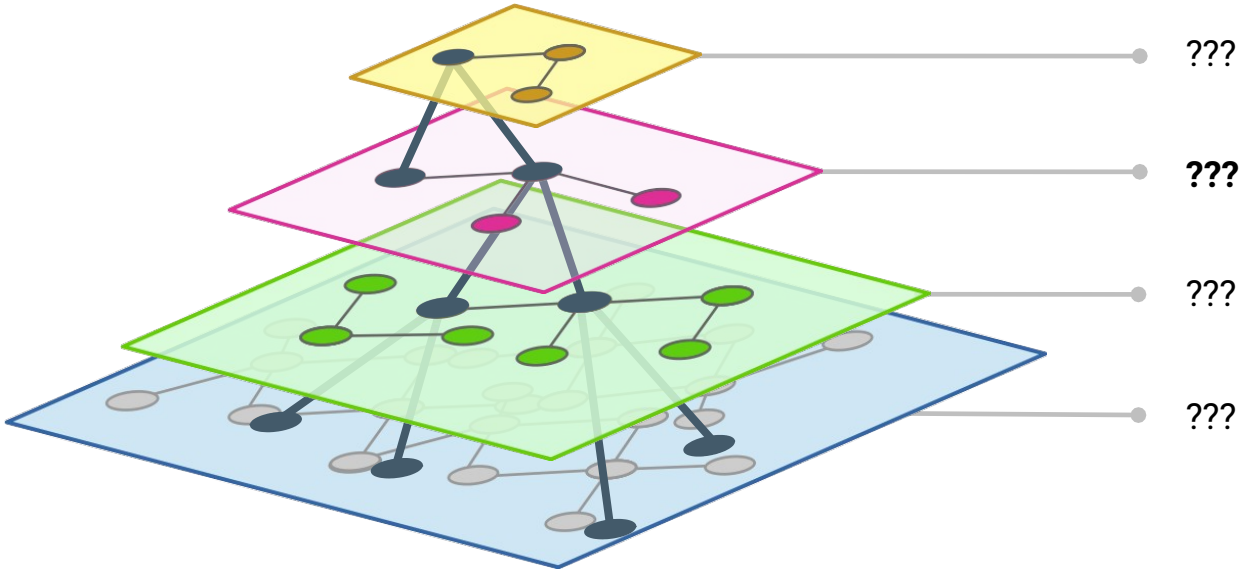
CVE-2016-6480 (Linux Kernel)

Race condition in the Linux Kernel in version 4.7 in `ioctl_send_fib` in `drivers/scsi/aacraid/commctrl.c`

https://joern.io/docs/kernelexamples/

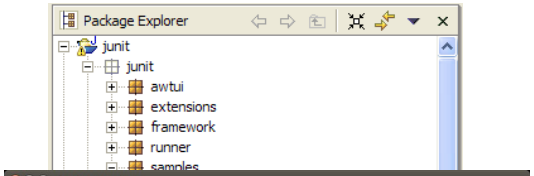
Let’s get technical

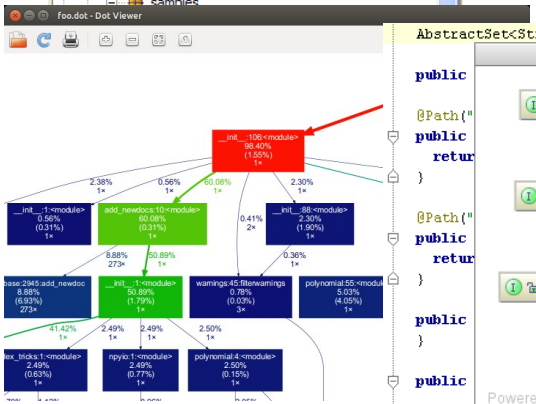
2

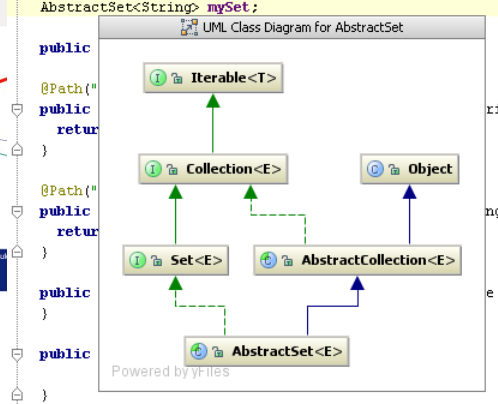


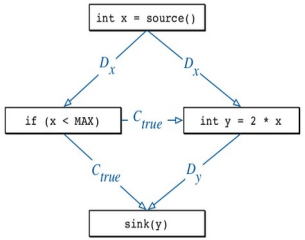
Low-level Graph Representations of Programs

- Each graph provides a different perspective on the code
- **Can we merge them?**

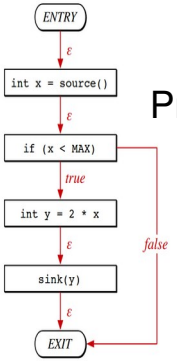


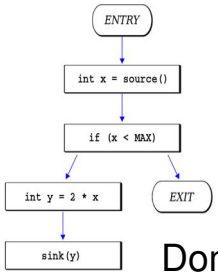






Program dependence graphs





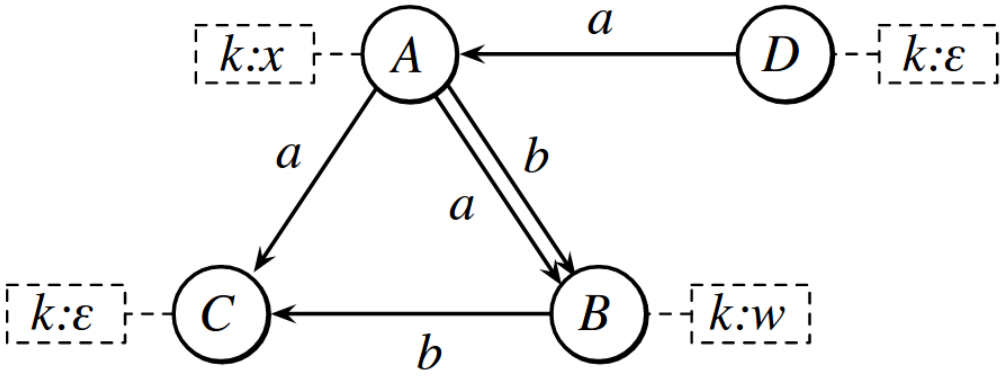
Dominator tree

ShiftLeft

Combining Graphs with “Property Graphs”

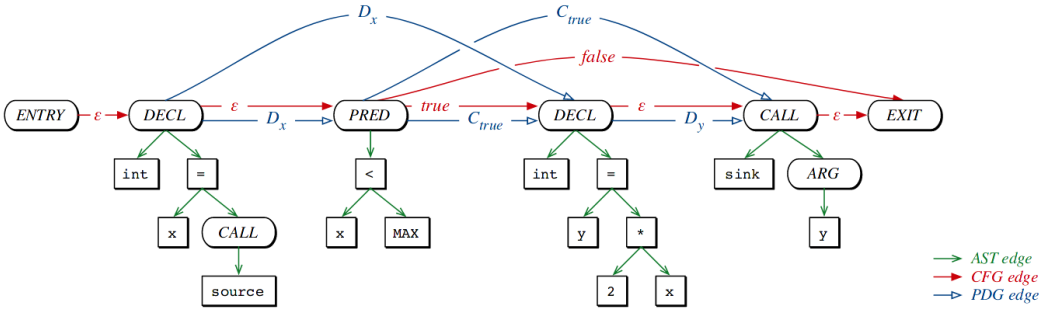
2

- *“A property graph is a directed edge-labeled, attributed multi-graph”*
- Attributes allow data to be stored in nodes/edges
- Edge labels allow different types of relations to be present in one graph



Modeling and Discovering Vulnerabilities with Code Property Graphs

Fabian Yamaguchi*, Nico Golde†, Daniel Arp* and Konrad Rieck*
*University of Göttingen, Germany
†Qualcomm Research Germany



Specification - Key Design Ideas

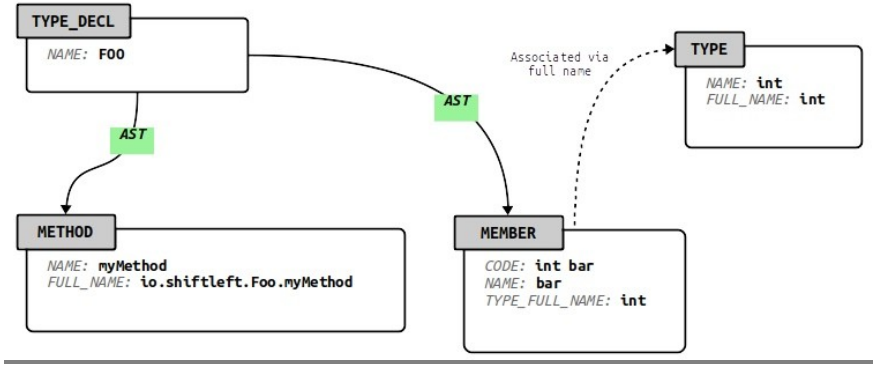
- Specification that works over programming languages
- Provide generic representation for core programming language concepts
  - Methods/Functions
  - Types
  - Namespaces
  - Instructions
  - Call sites

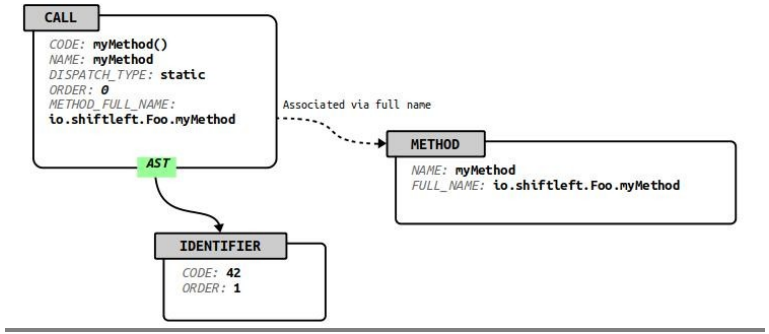
- Encode control flow structures only via a control flow graph
- Model only local program properties and leave global program representations for later analysis stages

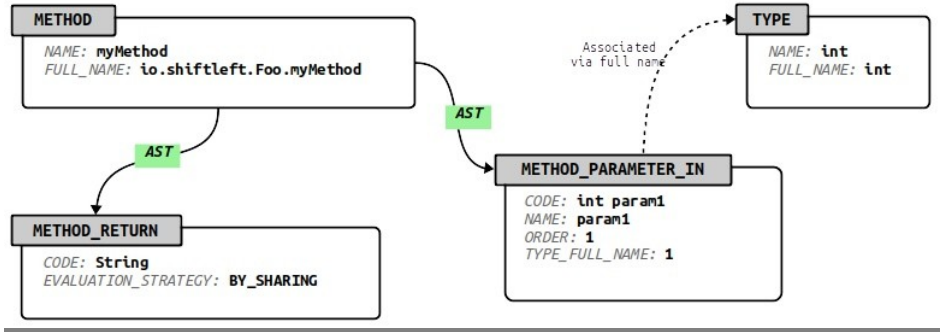
OSS Specification

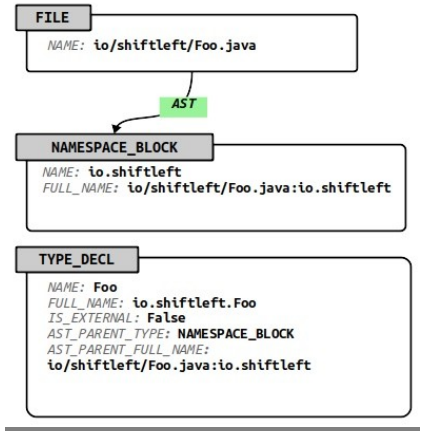
`github.com/ShiftLeftSecurity/codepropertygraph`

2



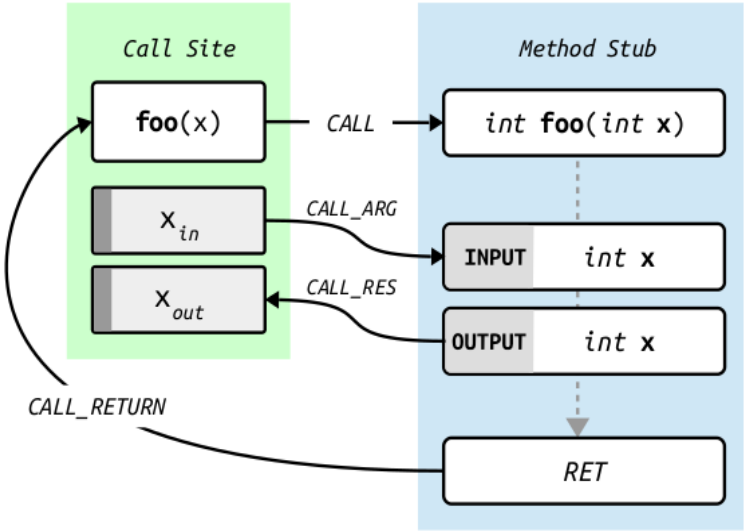






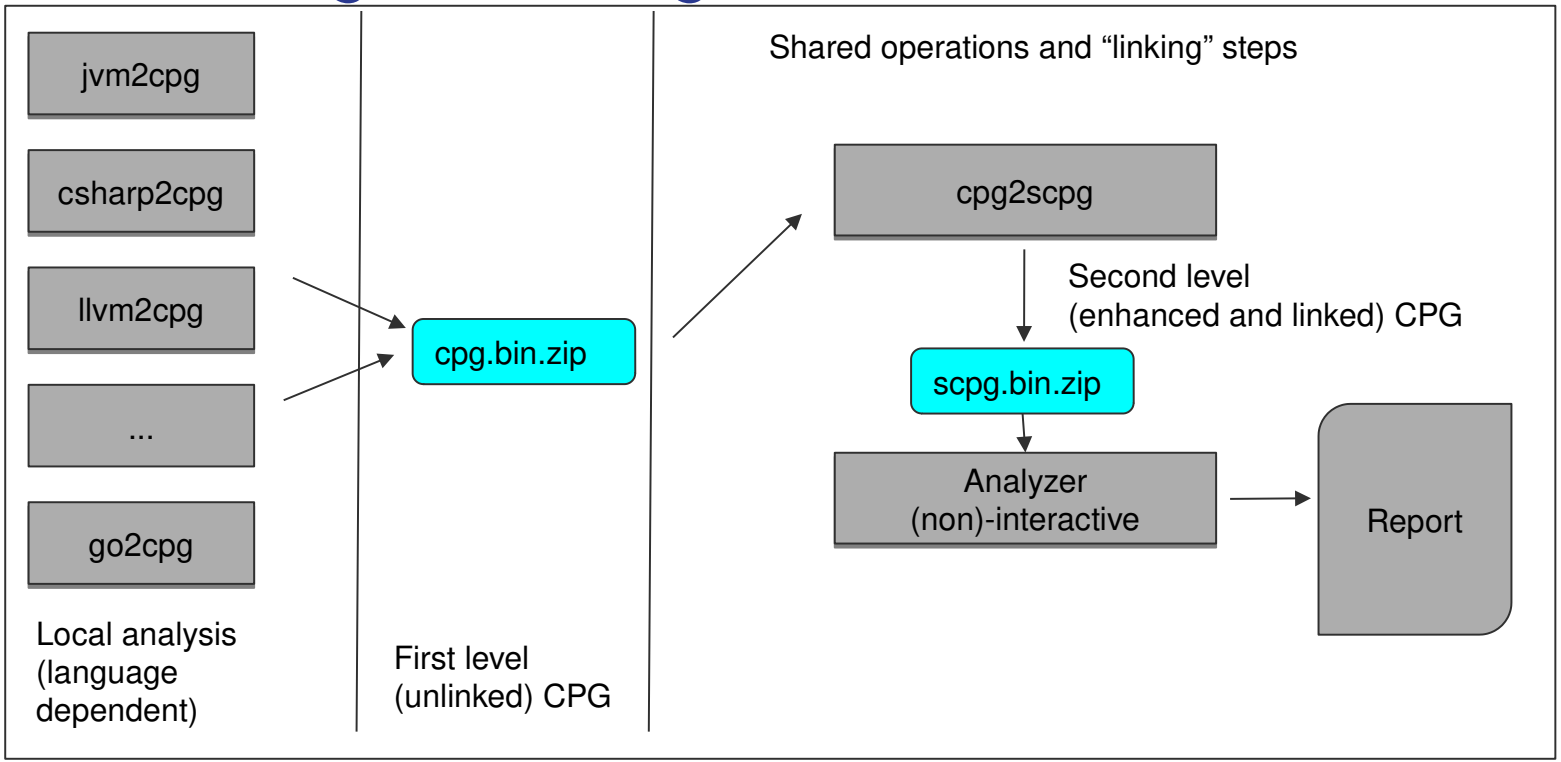
“Container” for Code over arbitrary Instruction Sets

- Define only a common format for representing code
- Allow arbitrary instruction set (given by semantics) as a parameter
- Represent all code using only
  - call sites and method stubs
  - call edges, and control flow edges
  - data-flow semantics via data flow edges



Second Stage: “linking”

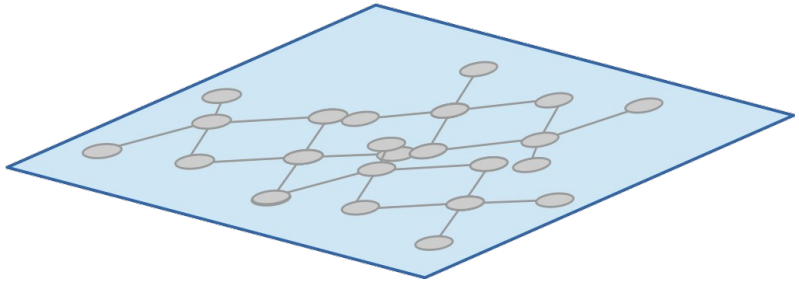
2



Base Layer of the Code Property Graph

2

- Production quality version of 2014 code property graph
- Language-independent intermediate representation of control-flow and data-flow semantics
- Inter-procedural, flow-sensitive, context-sensitive, field-sensitive data-flow tracker available that operates on this representation
- Heuristics and street smarts to terminate in < 10 minutes



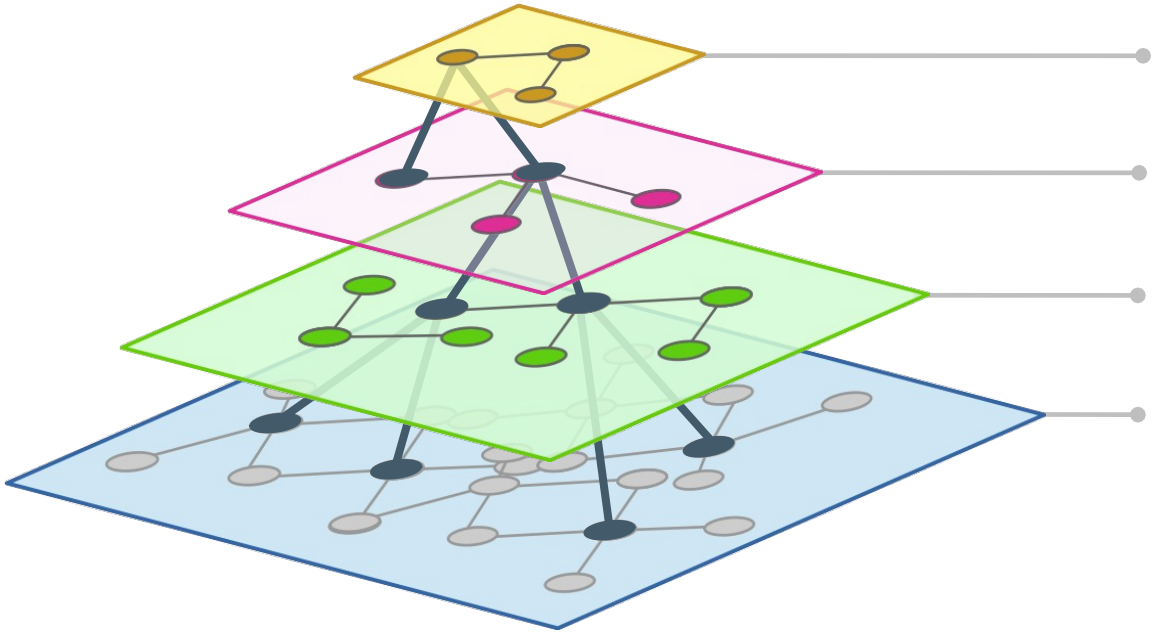
Outside Information, Business Logic, and False Positive (FP) Reduction

2

- Literature deals a lot with FPs due to model limitations
- In practice, most FPs result from context information, e.g., information about the business logic, that you cannot deduce from the code alone:
  - *“This is an internal service that only our admin uses”*
  - *“Without first convincing the authentication server, this code would never be executed”*
  - *“Due to $aliens, this integer is always 5 and thus cannot be negative”*
- Ability to model the *$aliens* part is crucial to reduce false positives
- **We do this mostly via passes that tag the graph**

Summary

2



Vulnerabilities

Multiple domain-specific layers

Call graph, type hierarchy, **data flows**, configurations, dependencies

Base layer - low level local program representations: syntax, control flow, methods, types.

Scaling Static Analysis

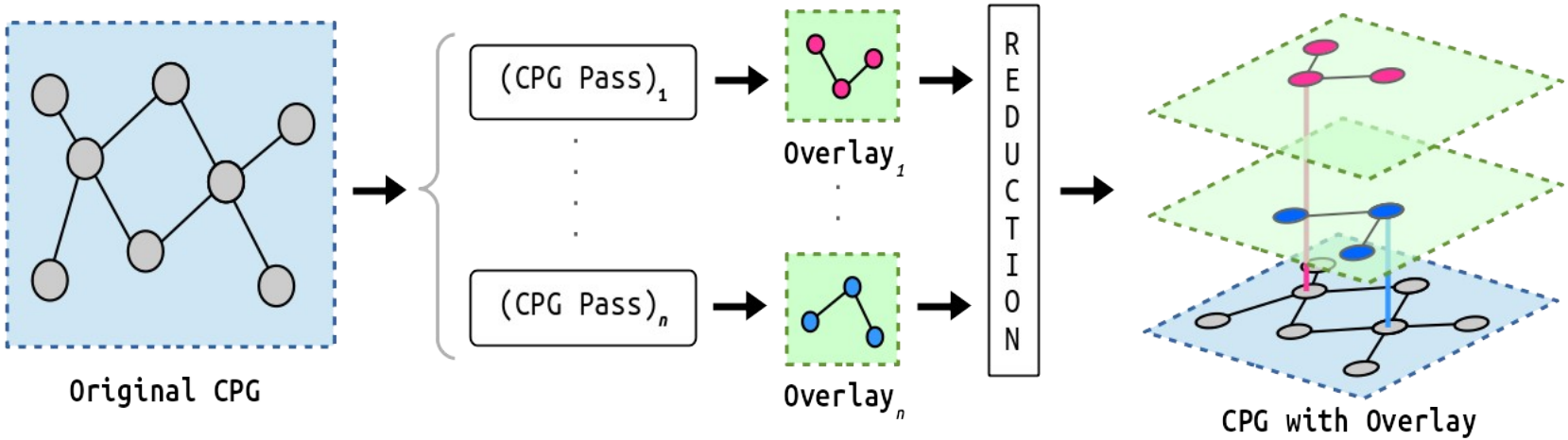
2

- Summaries
  - Scaling static analysis requires “summaries” of program behavior (in order to skip duplicate calculation of facts, e.g., for library methods)
  - Calculating summaries for data flow is common practice
  - **Upper layers of the CPG generalize the concept of a summary**

- Parallelism
  - Processors aren’t getting much faster, but you’re getting more and more cores.
  - Literature has very little to say about multiple cores, let alone multiple cloud instances
  - **CPG passes are a design with parallelism in mind**

Designed for Distributed Computing

- Passes can be run in a sequence like the passes of a compiler
- The design also allows to run independent passes in parallel though!



Ok ok… that was “interesting” ...

3

But what are you *actually* doing every day?

Technical Environment (*Codescience* Team)

3

Language:	
IDE:	mostly Intellij, some Vim, Sublime etc.
SCM:	git
Reviews / PRs, etc.:	GitHub
Buildtool:	sbt
CI/CD:	Jenkins, Grafana, Dockerhub, jFrog, Maven Central
Communication:	Mail, Slack, Zoom

Dev Process (*Codescience* Team)

3

Mostly sales-process-driven:

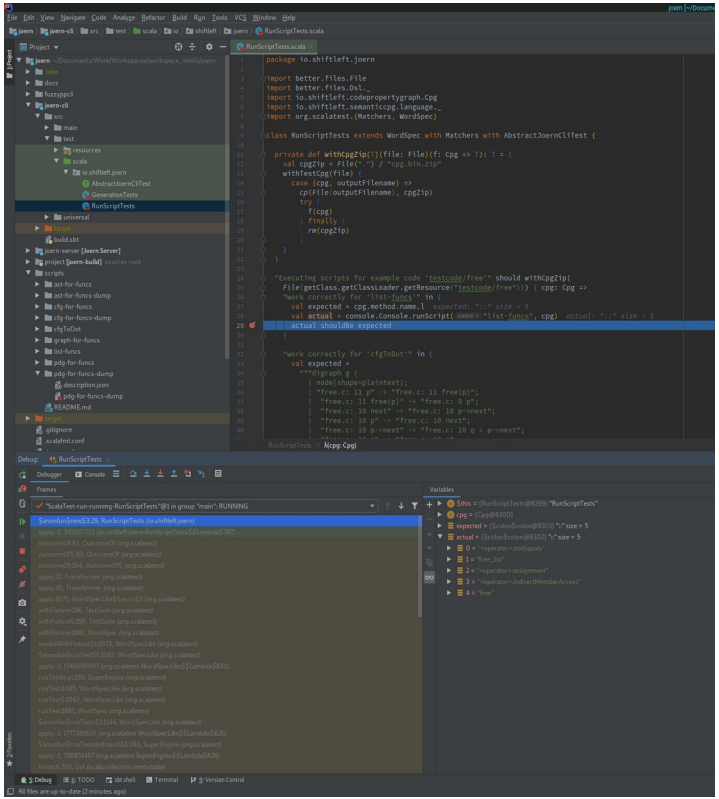
- Proof-of-Concept-oriented: potential new customers want to see *our* stuff working in *their* environment.

- Once they paid: mostly maintenance mode. Bug-fixing, ad hoc new features.

Master the Tooling

3

- SCM
- **Your IDE**
  - *Debugger*
  - *Shortcuts*
  - *Refactorings*
- Console-based stuff
- some scripting



Other Lessons learned …

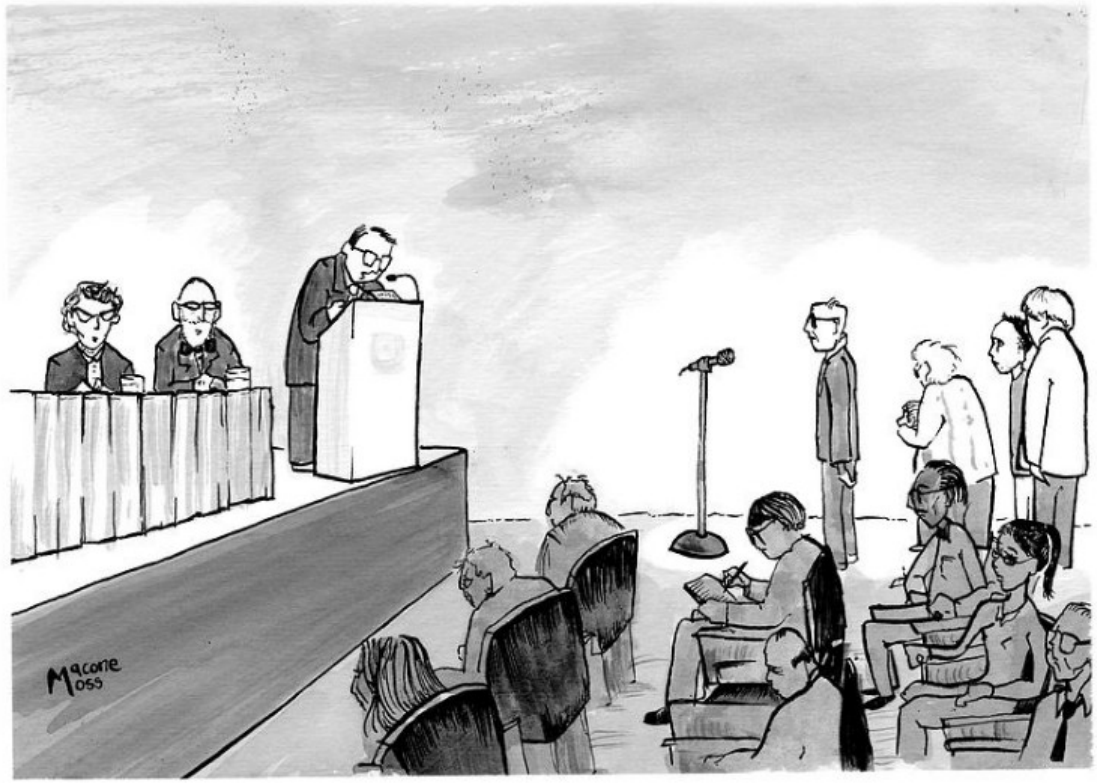
3

- It doesn’t have to be perfect — just “good enough”
- No-one knows everything
- You are responsible for your own learning path
- Don’t get overwhelmed
- Take a break

Other Lessons learned …

3

- What’s the most important language in programming?
- Talking to humans is way more important than talking to machines
- Have a deep understanding of what you are building and why
- If code review in your team is a stressful experience you are doing it wrong
- Something *will* go wrong, be prepared
- Don’t be afraid to say “I don’t know”
- Learn in public



"We'd now like to open the floor to shorter speeches disguised as questions."