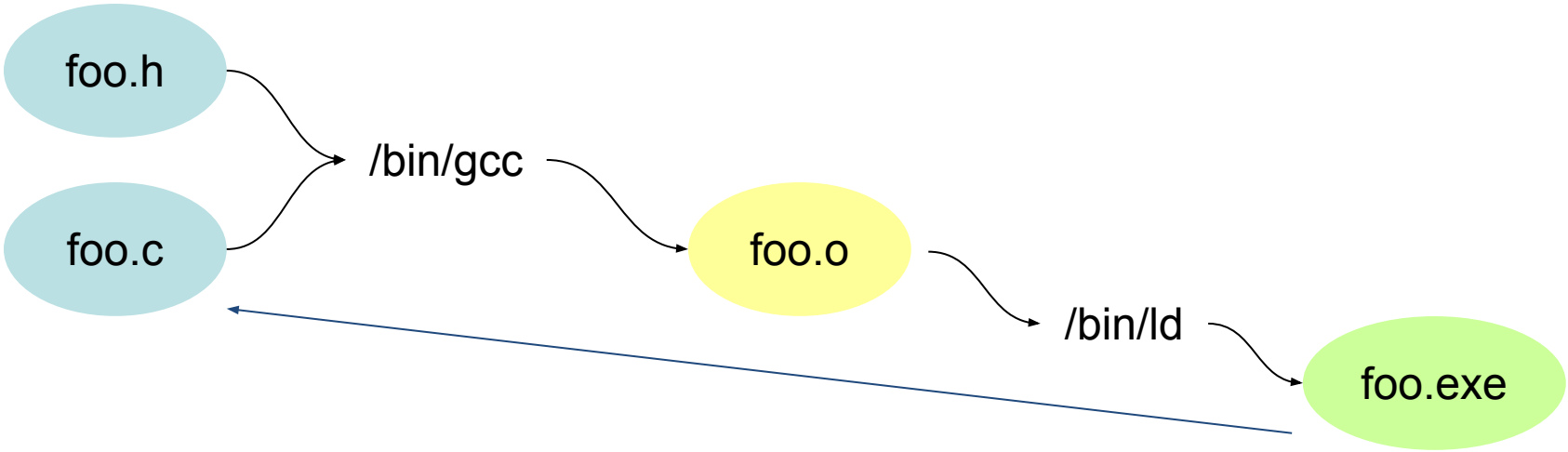
Debug your build by (s)tracing and reversing



nexB

Philippe Ombredanne

My mission: **make it easier to reuse FLOSS**

Enthusiast FLOSS developer
AboutCode, Linux kernel, a bit on strace, SPDX (and Eclipse, JBoss, and more)

CTO at nexB Inc. a software company helping software teams understand where their code comes from (and its licensing, vulnerability, quality, etc) with a combo of:

- FLOSS tools
- a commercial enterprise Dashboard

nexB

Why should you care?

Do you know what is in your binaries?

- Systems assembled from 1000's FOSS packages
- Multi-level complex build is like magic
- Packages fetched at build time
- Containers, custom Linux stacks, embedded
- Buggy, vulnerable packages and license issues

nexB

The problem

Given some binaries or package:

- **where do they come from?**
- **which known FOSS package are they built from?**

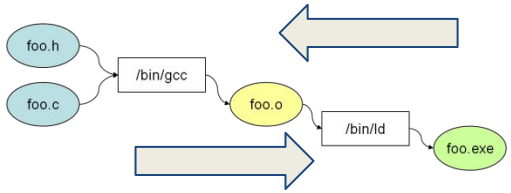
Complete and Corresponding Source code anyone?

Both in the large (whole Android stack)

or in the small (which .c files are in this ELF?)

Techniques to get there

- Dynamic **Forward** build tracing
  - **need to run the build**
  - **create graph from sources to binaries**
  - **query both ways**



Off topic:

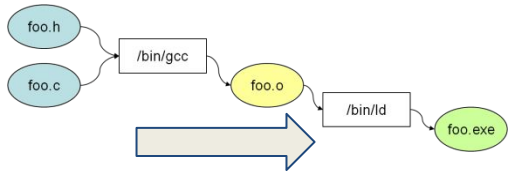
- compiler, build tool instrumentation
- backward, static, runtime execution tracing
- disassembly, emulation
- compiler conventions (e.g. Java), dependencies analysis
- signatures and fingerprints, shared string content
- build log analysis, symbols and debug symbols

nexB

Build tracing: the problem

- Static analysis is not an exact science
- Difficult to obtain debug artifacts: debug builds are not the norm
- Builds are complex and hard to modify
- And eventually impossible to instrument
- Hard to conclude that something is NOT built
- Built != Deployed in a software product

nexB



The ideal solution

Should be very easy on the developer

NO CHANGE needed to the build and config

Should provide 20/20 vision in analysis

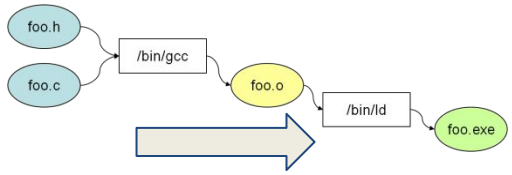
Should be 100% accurate

nexB

Syscalls to the rescue!

- The "**machine**" language of the Linux kernel
- 100% accurate

- Everything that touches a file, the network ends up in a syscall
- Very low level e.g.:
  - open/read/write file
  - spawn process and run executable



TraceCode

Run a build under “strace”, a system call tracer for Linux

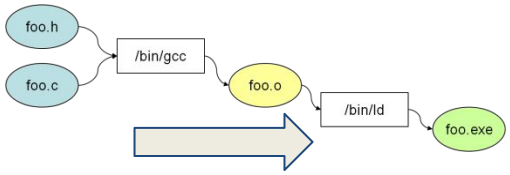
- Collect a trace of most EVERY syscalls

Process the trace

- ... and rebuild a **directed graph** of the file transformation that took place during the build

Query the graph for fun and profit!

nexB

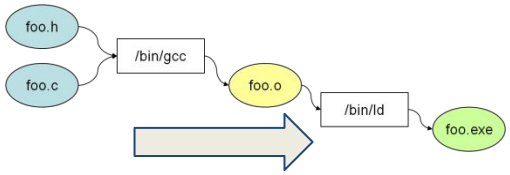


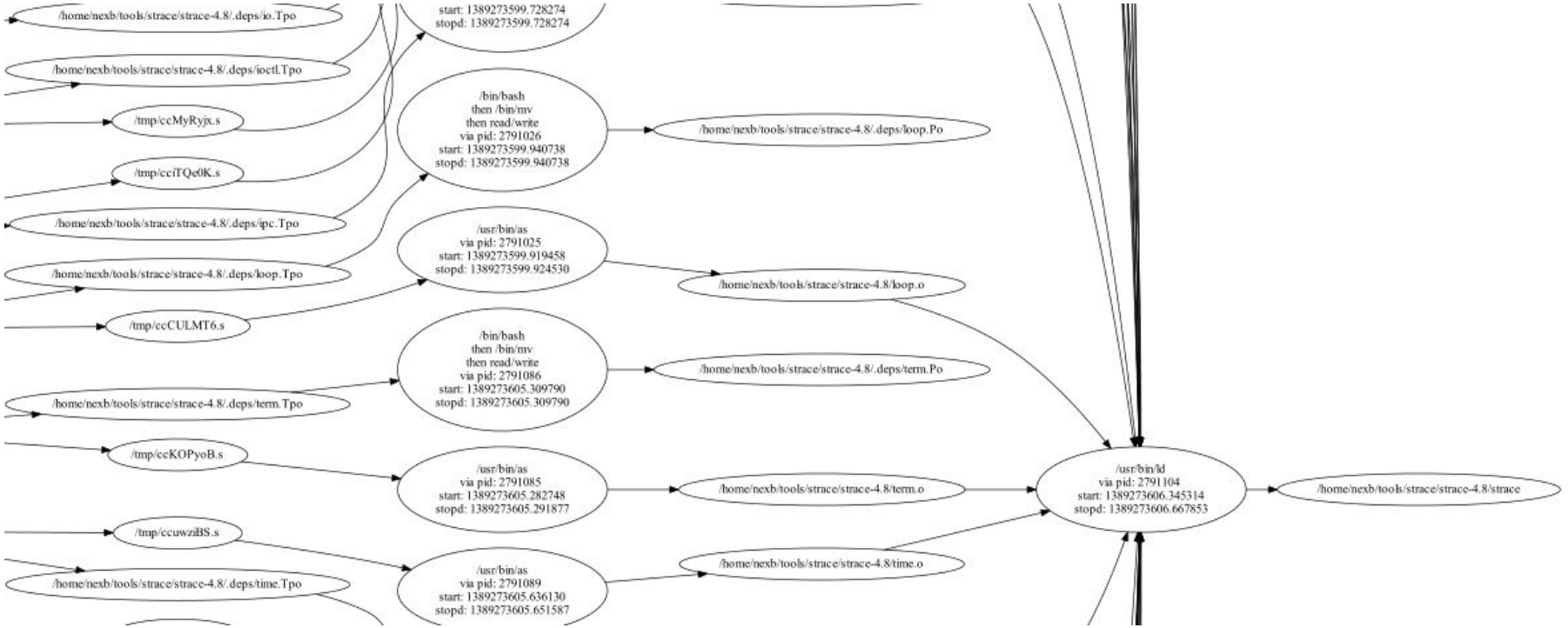
TraceCode (2)

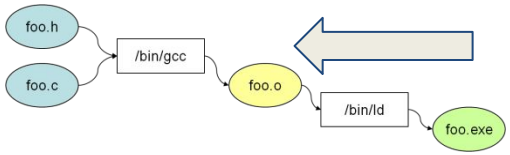
- Completely 100% **agnostic** wrt. compiler, toolchain or programming language
- Does not require *__ANY__* change to the build process
- Only need to run a traced build under strace
- Can provide 20/20 vision in the build process

nexB

TraceCode graph



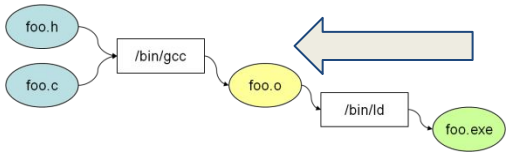




The models

- Executable
  - An exec is loaded in a process: we keep the command, args and time stamp
- ReadWrite
  - An atomic rename or copy-like operation with a src and a target path
- Operation
  - An operation, where a command in a process read sources and writes targets at some time stamp
- Process
  - Hold lists of file reads/writes/readwrites, list of execs and list of forked children.

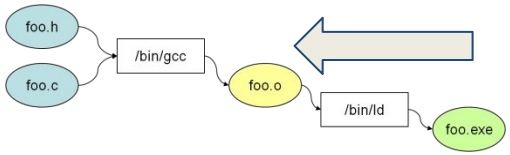
nexB



Complications...

- Parsing strace
- File descriptors lifecycle and recycling
  - limited life span
  - problematic for long running, single exe multi-file builds
  - ***Need to integrate an FD timeline?***
- Junk, temp and non interesting files
  - I lied: this is not 100% build-system agnostic
  - Temp/junk filtering requires a fine understanding
  - ***Create profiles for common build envs?***

nexB



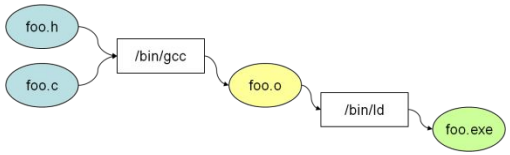
Beyond...

Knowing a binary provenance gets you a long way...

- But how do you know if this is vulnerable?
- But how do you know it's FLOSS license?

We need clarity in FLOSS licensing and vulnerabilities!

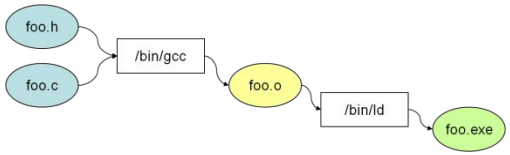
nexB



Tools

- TraceCode - build tracing
  - https://github.com/nexB/tracecode-toolkit
  - Written in Python, Apache-licensed.
- next steps
  - FD timeline
  - static analysis and static reversing
  - binary signatures

nexB

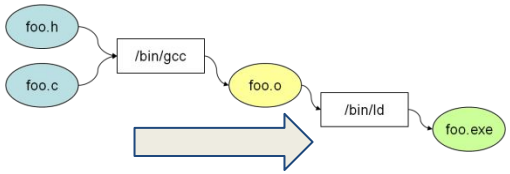


I lied again!

"As a bonus, I will also present new FOSS extension that perform similar build reversing but using static analysis only."

- Still a work in progress
- Only some bits for ELF/PE/Mach-O symbols & debug symbols are there
- https://github.com/nexB/scancode-toolkit-contrib

nexB



Credits

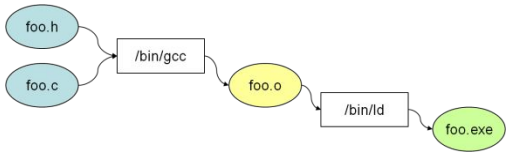
strace rocks! ldv should in the room!

TraceCode is implementing this paper:

"Discovering Software License Constraints:
Identifying a Binary's Sources by Tracing Build Processes"

By Sander van der Burg, Julius Davies, Eelco Dolstra, Daniel M. German, Armijn Hemel.
http://www.st.ewi.tudelft.nl/~sander/pdf/publications/TUD-SERG-2012-010.pdf

nexB



Plug

Join me for

Meet purl: a "mostly" universal software package URL that purrs.

16:00 in the Package management dev room

and:

Package Management Panel Discussion

17:00 in the Package management dev room

nexB

Thank you!

Credits

Special thanks to all the people who made and released these awesome free resources:

- Presentation template by SlidesCarnival
- Photographs by Unsplash
- Icons from openclipart.org

And all the FLOSS software authors!