A Suite of Tools for Pragmatic Program Analysis of C and C++
Daniel S. Wilkerson

• Build Interceptor
• Elsa/Oink
• Delta
Acknowledgments

• Alex Aiken and David Wagner funded and supported the project
• Scott McPeak wrote Elsa, extended Delta, collaborated extensively on thinking, debugging, learning/“puzzling out” C++
• Jeff Foster and Rob Johnson provided Cqual
• Build Interceptor extends work by Ben Liblit and Hao Chen and Geoff Morrison
• Ben helped substantially with C++ also
• Many others helped with ideas, testing, a little implementation, etc.
Helped with Talk

- Matt Harren
- Simon Goldsmith
- Adam Chlipala
- Tachio Terauchi
- Scott McPeak
Analyzing Code In the Wild

• Much real code is C/C++
• Messy/unique build process
• Hard to isolate failure-inducing part of large inputs (Kernel, Mozilla)
• Want to re-use the front-end
• Want analyses to compose
  – Poor-man's flow-sensitive cqual: run control-flow, then data-flow (repeat?)
Outline of process

- Build as usual; use **Build Interceptor** to get .i files (post-preprocessed files)
- Use **Elsa/Oink** to parse and typecheck C/C++
- Debug minimizing large inputs with **Delta**
- Ship your analysis with Oink so others can re-use it.
Where we are

- Build Interceptor
- Elsa/Oink
- Delta
Build Interceptor: features

• Captures .i files generated during build
  – and linking info for whole-program analysis
• No need to modify build process!
  – Scales to hundreds of projects
• 4th generation, builds on work done in Hao Chen's and Ben Liblit's projects
  – 92.5% of Red Hat 7.3 projects *
  – Being improved further for Debian
Build Interceptor, p.2: usage

• Works by replacing system tools
  – You must have root
  – Prevents stupid mistakes with checks and helpful on/off make targets

• Result is
  – a .i file for every compile
  – a .ld file for every link
    • The list of .i files that went into the ELF

• To get whole-program analysis just pass the .ld file to Oink with a flag.
Build Interceptor, p.3: how it works

- When gcc runs cc1, gets our script instead, which
  - copies .i file input
  - Runs real cc1
  - Appends assembly data string in an unused ELF section containing name of copied .i file

- This section concatenates during linking and survives stripping
Build Interceptor, p.4: how it works diagram

foo.c → cpp → foo.i → ccl_int

bar.o → ld_int → ld

objdump .note.ccl → "/tmp/foo.i", "/tmp/bar.i"

objdump .note.ld → “foo.o, bar.o”

foo.s
.section .note.ccl
.ascii "/tmp/foo.i"

as → foo.o

foo.exe
.note.ccl: "/tmp/foo.i"
.note.ld: “foo.o, bar.o”

objcopy
Build Interceptor, p.5: you can really control build

- We also intercept `collect2/ld`
  - With `-t`, get list of `.o` files as they are linked
  - These are stored into ELF file using `objcopy` to insert an unused ELF section
  - This trick due to Hao Chen and Ben Liblit

- Intercept `make` to turn off `-j`
- Intercept `cpp` to turn off `-P`
- Provide our own `gcc` spec file, etc.
Build Interceptor, p.6: already run for you

- Red Hat 7.3 .i files are available if you want to avoid this for now
  - We also give list of the subset of those that Elsa/Oink can parse
  - On those, any bugs are yours, not Elsa/Oink
  - We forgot to include .so files in whole-program lists
    - Oops
    - We'll fix that
Where we are

- **Build Interceptor**
- **Elsa/Oink**
- **Delta**
Scott McPeak's Elsa

- Parses, typechecks C
- & simplifies C++
  - Result is simplified down to “Java with Multiple inheritance”
  - Pretty clean design internally
- Red Hat 7.3
  - >99% C files go through
  - >50% C++ files go through with old headers
- Kernel, Mozilla and Qt go through
Elsa, p.2: “Java”-semantic output

• Instantiates templates
• Turns operator overloading into function calls
• Inserts implicit function calls: ctors, dtors.
• Writes implicit code: default ctor, dtor, operator =()
• Looks up all variables for you
Where we are

- **Build Interceptor**
- **Elsa/Oink**
- **Delta**
Oink

• Client of Elsa
  – Which provides intra-procedural control-flow for Oink

• Generic flow-insensitive data-flow analysis
  – Client: Jeff Foster's monomorphic Cqual
  – Rob Johnson's polymorphic in the works

• Linker imitator
  – Order of linking probably right by default
Oink, p.2: compose analyses

• Designed to allow analyses to collaborate/compose if they
  – Step 1: annotate types and AST (using the generic annotation mechanism)
  – Step 2: then make conclusions

• Can run step 1 for many analyses before running a custom step 2

• Can ship your analyses with Oink
  – Can also ship combo analyses built from others
Oink, p.3: projects using it

• “Scrash”, Matt, Naveen et. al.: “eliminate sensitive information ... [in] crash report”
  - Annotated with $tainted, flowed qualifiers, Pretty printed back out

• Umesh: “Find sensitive data that is live at the time of fork()”
  - Combo data/control-flow analysis
  - Simply added usual function call edges to intra-procedural control-flow
  - Something like 1 month to get it to work with lots of help from me
Oink, p.4: work in progress

• Karl Chen and I working on verifying Debian C code has no format-string bugs
  – Goal: become part of Debian vetting process
  – adding other analyses would then have low marginal cost

• Hao started porting MOPS to Oink, but became a professor; said student is doing it

• Zhendong & Alex said wanted it for something

• Could add Wes Weimer's Java parser
  – Typecheck into C++ AST and Types
  – Any C++ analysis is also a Java analysis
Where we are

- **Build Interceptor**
- **Elsa/Oink**
- Delta
Delta

• Minimizes interesting files
  – Such as those that crash your program
• Just provide a test of interestingness
  – Such as: `grep 'error'`
• Used on quarter-million line inputs
  – Seems to always stop at a page or two
• Implementation of “Delta Debugging” algorithm from Saarland University
  – My code is easier to read than the paper
Delta, p.2: how it works

- Simulated annealing
- For i going down (temperature)
  - For each group of $2^i$ lines
    - Remove group and run test
    - Deletion permanent if test passes
- Original algorithm also tries negative: deleting “all but this group”
  - this is a waste of time
- “Dumb as hell but goes like 60”
  - Feynman
Delta, p.3: using language structure

- Guesses much better if we use the language structure somehow
- Filter `topformflat` added by Scott
  - Omits newlines > a given nesting depth
  - Depth=0 means one line=whole function
  - Minimize at each depth with Delta, with depth increasing from 0
- Result: Delta “understands” C-like syntax nesting
Delta, p.4: results

- Don't know if Elsa/Oink possible without it
- Alex: "I used it for one assignment where the students were given something that we knew would minimize pretty well. They all liked it."
- User Ed Avis: "successfully used delta to track down . . . perl bugs"
- Works on configuration files, etc.
Summary

- Use the tools to analyze real programs
- Write analyses with Oink and they will all compose together!
  - Saves duplicated effort on front-ends
  - Makes composite analyses possible
- If Elsa/Oink achieves critical mass, all analyses for C/C++ will be written in it and will compose with one another
- Take over the analysis universe.
- Feel free to talk to me about using any of these tools for a demo/lesson.