Symbolic execution systems

Resurgence

- Two key systems that triggered revival of this topic:
- **DART** — Godefroid and Sen, PLDI 2005
  - Godefroid = model checking, formal systems background
- **EXE** — Cadar, Ganesh, Pawlowski, Dill, and Engler, CCS 2006
  - Ganesh and Dill = SMT solver called STP (used in implementation), Cadar and Engler = systems
- Now on to next-generation systems

SAGE

- **Concolic executor** developed at **Microsoft Research**
  - Grew out of Godefroid’s work on DART
  - Uses generational search
- Primarily **targets bugs in file parsers**
  - E.g., JPEG, DOCX, PPT, etc
  - Good fit for concolic execution
    - Likely to terminate
    - Just input/output behavior

SAGE Impact

- **Used on production software at MS**. Since 2007:
  - 500+ machine years (in largest fuzzing lab in the world)
    - Large cluster of machines continually running SAGE
  - 3.4 Billion+ constraints (largest SMT solver usage ever!)
  - 100s of apps, 100s of bugs (missed by everything else…)
    - Ex: *1/3 of all Win7 WEX security bugs found by SAGE*
  - Bug fixes shipped quietly to 1 Billion+ PCs
  - Millions of dollars saved (for Microsoft and the world)
  - SAGE is now used daily in Windows, Office, etc.

http://research.microsoft.com/en-us/um/people/pg/public_psfiles/SAGE-in-1slide-for-PLDI2013.pdf

KLEE

- **Symbolically executes LLVM bitcode**
  - LLVM compiles source file to .bc file
  - KLEE runs the .bc file
  - Grew out of work on EXE

- Works in the style of our basic symbolic executor
  - Uses `fork()` to manage multiple states
  - Employs a variety of search strategies
    - Primarily **random path + coverage-guided**
  - Mocks up the environment to deal with system calls, file accesses, etc.

- **Freely available with LLVM distribution**

KLEE: Coverage for Coreutils

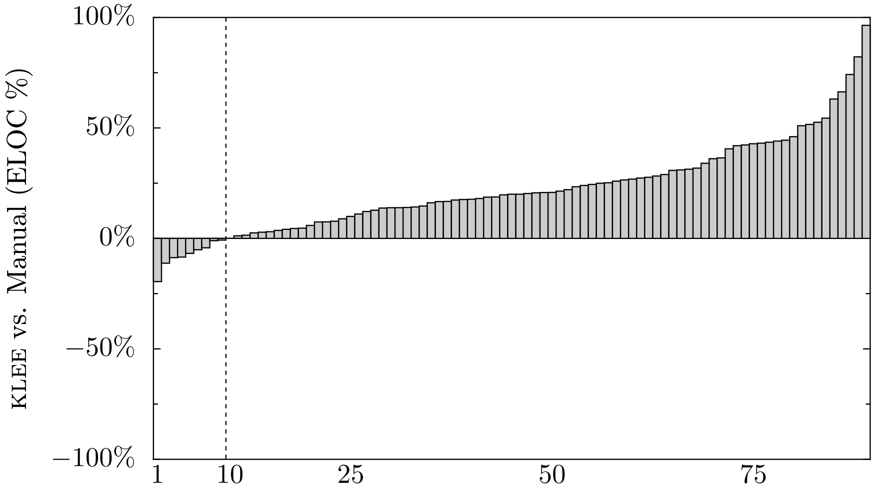
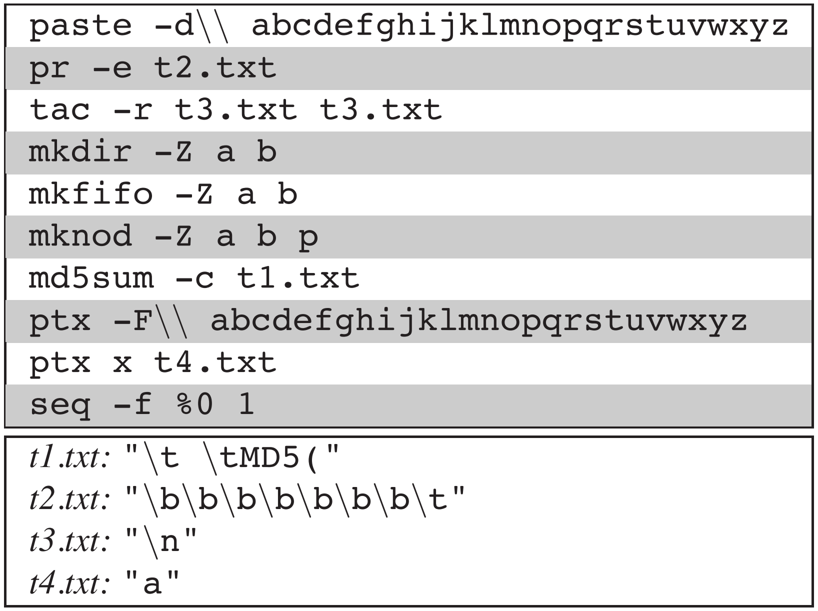


Figure 6: Relative coverage difference between KLEE and the COREUTILS manual test suite, computed by subtracting the executable lines of code covered by manual tests (L_{man}) from KLEE tests (L_{klee}) and dividing by the total possible: $(L_{klee} - L_{man})/L_{total}$. Higher bars are better for KLEE, which beats manual testing on all but 9 applications, often significantly.

Cadar, Dunbar, and Engler. KLEE: Unassisted and Automatic Generation of High-Coverage Tests for Complex Systems Programs, OSDI 2008

KLEE: Coreutils crashes



```
paste -d\\ abcdefghijklmnopqrstuvwxyz
pr -e t2.txt
tac -r t3.txt t3.txt
mkdir -Z a b
mkfifo -Z a b
mknod -Z a b p
md5sum -c t1.txt
ptx -F\\ abcdefghijklmnopqrstuvwxyz
ptx x t4.txt
seq -f %0 1
```

t1.txt: `"\t \tMD5("`
t2.txt: `"\b\b\b\b\b\b\b\t"`
t3.txt: `"\n"`
t4.txt: `"a"`

Figure 7: KLEE-generated command lines and inputs (modified for readability) that cause program crashes in COREUTILS version 6.10 when run on Fedora Core 7 with SELinux on a Pentium machine.

Cadar, Dunbar, and Engler. KLEE: Unassisted and Automatic Generation of High-Coverage Tests for Complex Systems Programs, OSDI 2008

Mayhem

- Developed at CMU (Brumley et al), **runs on binaries**
- Uses BFS-style search and native execution
  - **Combines best of symbolic and concolic strategies**
- **Automatically generates exploits** when bugs found

Mergepoint

- Extends Mayhem with a technique called **veritesting**
  - ***Combines*** **symbolic execution** with **static analysis**
  - Use static analysis for complete code blocks
  - Use symbolic execution for hard-to-analyze parts
    - Loops (how many times will it run?), complex pointer arithmetic, system calls
- Better **balance** of time **between solver and executor**
  - **Finds bugs faster**
  - **Covers more of the program** in the same time
- Found 11,687 bugs in 4,379 distinct applications in a Linux distribution
  - Including new bugs in highly tested code

Other symbolic executors

- **Cloud9** — Parallel, multi-threaded symbolic execution
    - Extends KLEE (available)
- **jCUTE**, **Java PathFinder** — symbolic execution for Java (available)
- **Bitblaze** — Binary analysis framework (available)
- **Otter** — directed symbolic execution for C (available)
    - Give the tool a line number, and it try to generate a test case to get there
- **Pex** — symbolic execution for .NET

Summary

- **Symbolic execution generalizes testing**
  - Uses static analysis to direct generation of tests that cover different program paths
- Used in practice to find **security-critical bugs** in **production code**
  - SAGE at Microsoft
  - Mergepoint for Linux
- **Many tools freely available**