Program analysis for security: Making it scale

David Wagner
U.C. Berkeley

Work by Hao Chen, Karl Chen, Rob Johnson, Ben Schwarz, and Jeremy Lin, Geoff Morrison, David Schultz, Jacob West

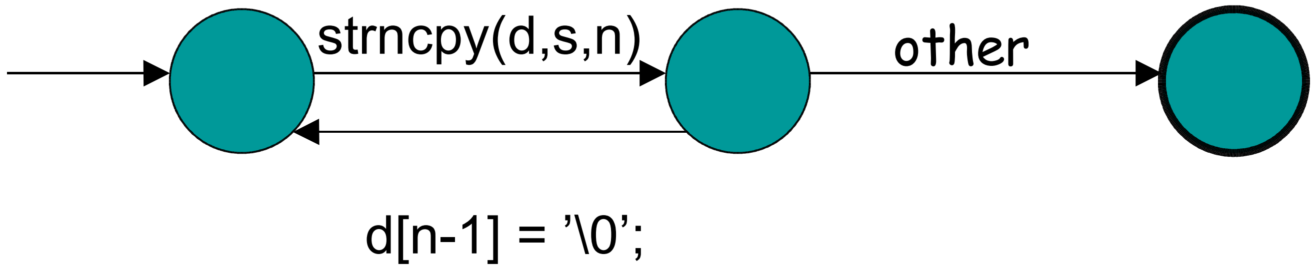
Outline

- Wrapping up the MOPS project
  - End-of-project experimental evaluation
  - Lessons

- Verification of security properties via type inference
  - Modular analysis
  - Preliminary results: user/kernel, format strings

Refresher on MOPS

- Pushdown model checking of C source code
- Security properties expressed as finite state automata



Example: A simple FSA to detect misuse of strncpy( ).
Error state indicates possible failure to null-terminate d.

(Real property is much more complex: many ways to terminate; pre-termination vs. post-termination; delayed termination.)

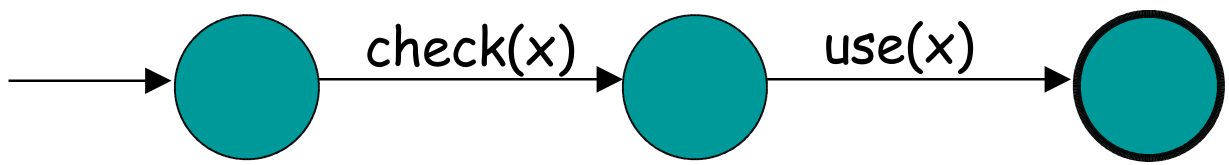
TOCTTOU (time-of-check to time-of-use)

- Canonical example of a TOCTTOU vulnerability:

```
if (access(pathname, R_OK) == 0)
        fd = open(pathname, O_RDONLY);
```

- Notice: not an atomic operation!

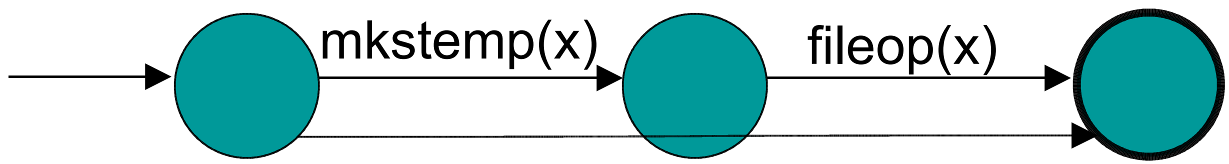
- Bug: Permissions may change between access() & open()
  - Attacker can arrange for this to happen in an attack



check = { access, lstat, stat, readlink, statfs }
use = { chmod, open, remove, unlink, mount, link, mkdir, rmdir … }

Insecure temporary file creation/use

- Temporary file creation requires special care:
  1) unguessable filename; 2) safe permissions;
  3) file ops should use fd, not filename (TOCTTOU)



{ tmpnam(), tempnam(), mktemp(), tmpfile() }

fileop(x) = { open(x), chmod(x), remove(x), unlink(x) … }

MOPS in the large

- Experiment: Analyze an entire Linux distribution
  - Redhat 9, all C packages (732 pkgs, ~ 50 MLOC)
  - Security analysis at an unprecedented scale
- Team of 4 manually examined 900+ warnings
  - 1 grad student; 3 undergrads new to MOPS
  - Exhaustive analysis of TOCTTOU, tmpfile, others; statistical sampling of strncpy
  - Laborious: multiple person-months of effort
- Found **79** new security holes in Linux apps

Security Property	Warnings	Real bugs	Bug ratio
TOCTTOU	790	41	5%
temporary files	108	34	35%
strncpy	668	(unknown)	~ 5-10%
Total	1597	79+	

Lessons & surprises from the MOPS effort

- Unexpectedly, most real bugs were local
- False alarm rate high. Doing better requires deeper modeling of OS/filesystem semantics.
  - Path sensitivity only good for $\leq$ 2x improvement
  - Many non-bugs were still very interesting (represented fragile assumptions about environment)
- Engineering for analysis at scale is highly non-trivial
  - Good UI, explanation of errors is critical
  - Build integration so important — and so hard — that we re-implemented it no less than four times
- But worth it: Large-scale experiments incredibly valuable
- Tech. transfer: techniques being adopted in commercial security code scanning tools

Bug #1: “zip”

Pathname from cmd line

```
d_exists = (lstat(d, &t) == 0);
if (d_exists) {
    /* respect existing soft and hard links! */
    if (t.st_nlink > 1 ||
                (t.st_mode & S_IFMT) == S_IFLNK)
        copy = 1;
    else if (unlink(d))
        return ZE_CREAT;
}
```

... eventually writes new zipfile to `d` ...

Bug #2: "ar"

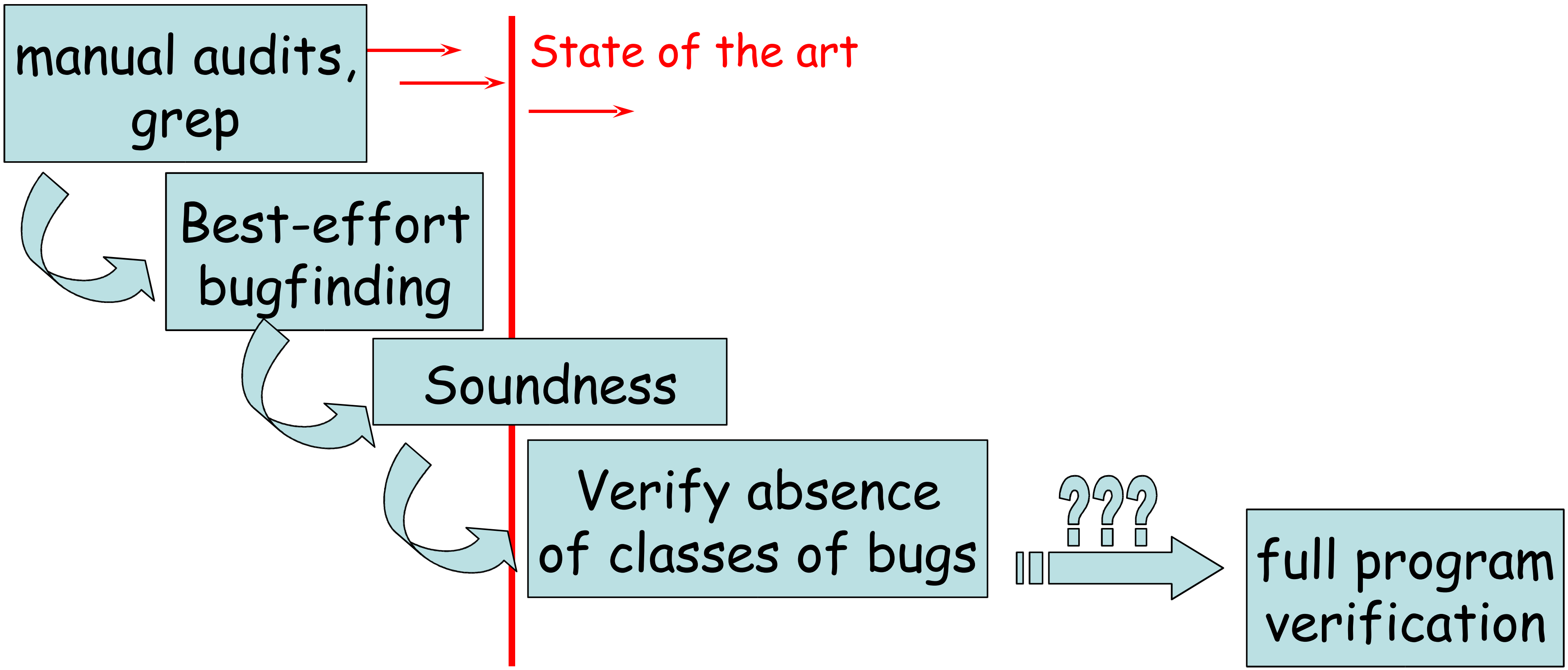
```
exists = lstat (to, &s) == 0;
if (! exists ||
     (!S_ISLNK (s.st_mode) && s.st_nlink == 1)){
   ret = rename (from, to);
   if (ret == 0) {
     if (exists) {
       chmod (to, s.st_mode & 0777);
       if (chown (to, s.st_uid, s.st_gid) >= 0)
          chmod (to, s.st_mode & 07777);
     }
   }
}
```

Bug #3

```
static void open_files() {
  int fd;
  create_file_names();
  if (input_file == 0) {
    input_file = fopen(input_file_name, "r");
    if (input_file == 0)
      open_error(input_file_name);
    fd = mkstemp(action_file_name);
    if (fd < 0 || (action_file =
                  fdopen(fd, "w")) == NULL) {
      if (fd >= 0)
        close(fd);
      open_error(action_file_name);
    }
}
void open_error(char *f) {
  perror(f); unlink(action_file_name); exit(1);
}
```

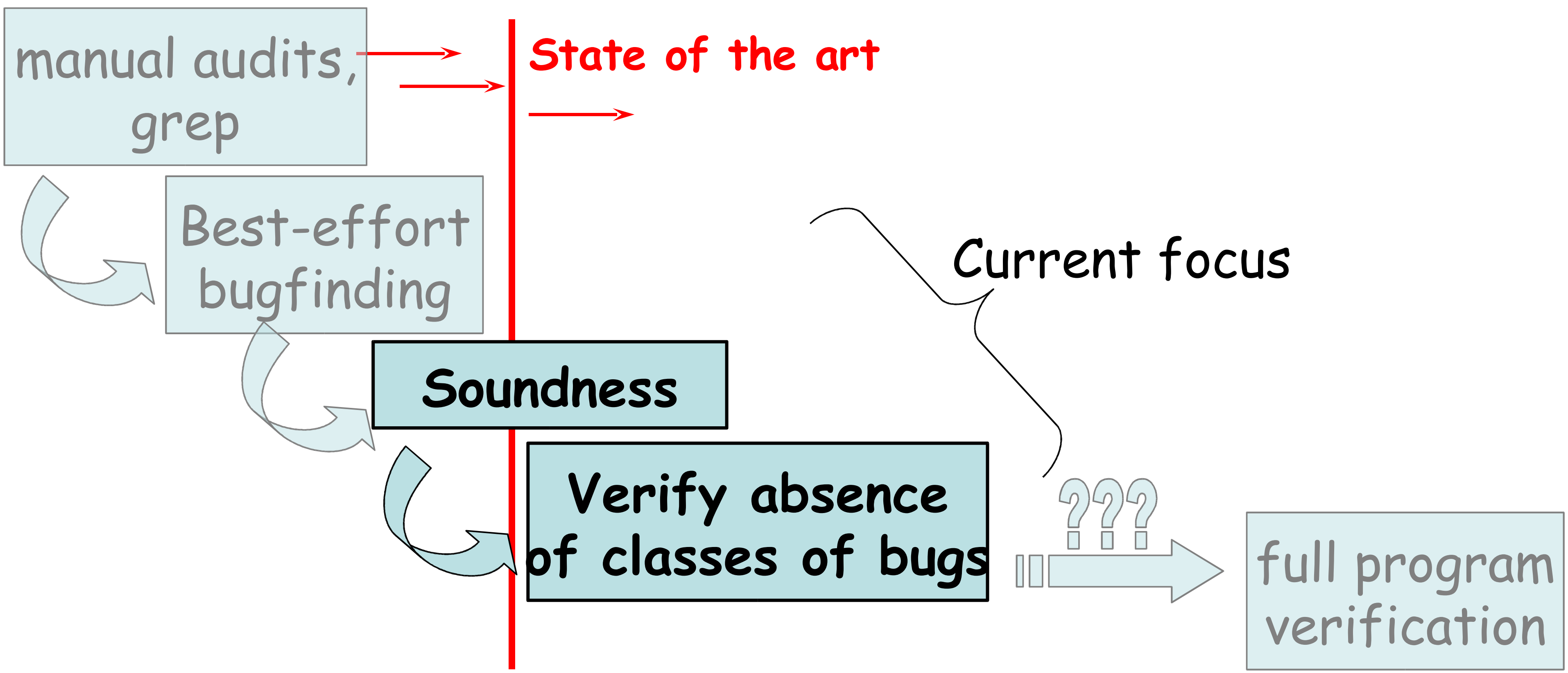
Current research

- Research direction: verify absence of data-driven attacks, using type inference



Current research

- Research direction: verify absence of data-driven attacks, using type inference



Input validation

- Q: Why is writing secure code hard?
  A: Secure programs must handle untrusted data securely, and must get it right *every single time.*

- Focus area: input validation
  - Untrusted data should be sanitized before it is used at any trusting consumer
  - Defends against data-driven attacks

- Strategy: Help programmers get it right "every time" with tool support

Why focus on verification?

- Previous work has studied best-effort bugfinding
  - Useful, but misses many bugs
- Challenge: verifying absence of (certain kinds of) bugs

- Verification has many benefits
  - For developers: (1) prevents shipping insecure code; (2) integration into build & QA process fixes bugs early (like regression testing)
  - For users: provides a security metric
  - Also, in our experience, verification finds more bugs

Refresher: user/kernel security holes

- Experiment: Can CQual verify absence of u/k bugs?
  - Sound whole-kernel analysis

Linux kernel	Warnings	Bugs	Annotations	Size
2.4.23-default	53	10	287	300K LoC

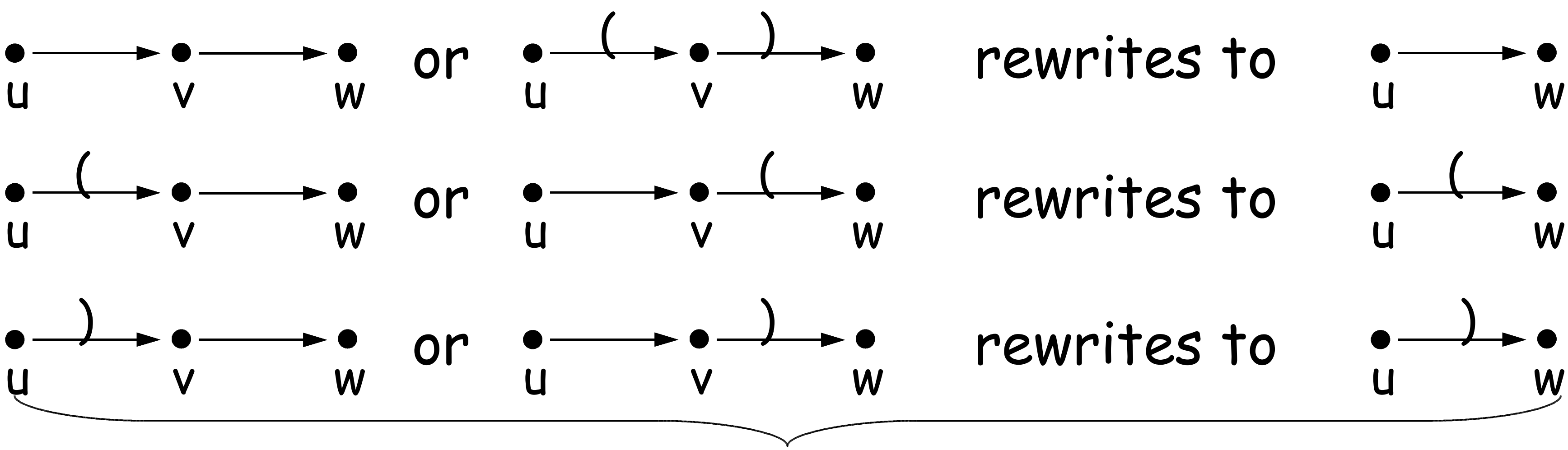
- Found 10 exploitable holes in Linux 2.4.23 core
  - Sparse: missed all 10 bugs; 7000 annotations; many FPs
  - MECA: missed 6/8 bugs; 75 annotations; very few FPs
  - Lesson: Soundness matters!
- Cost: 90 min. CPU time, 10GB RAM on 800MHz Itanium

- Conclusion: Memory usage is a key challenge for scalability

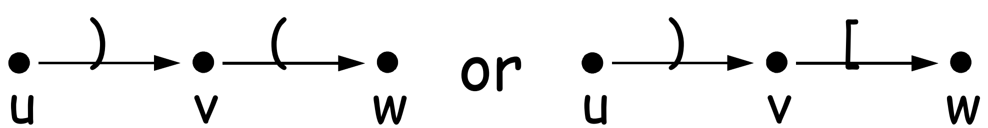
New: Modular type inference

- Reduce space complexity of CQual's CFL reachability analysis, by generating summaries for each module:

```
for (f in source-files)
    read f; minimize CFL graph by rewrite rules; store graph
read all graphs, & link together; perform CFL reachability
```



If v has local scope, rewrite & delete v (unless ineligible — see below)



v ineligible for deletion

Preliminary experiments: Format string holes

- Experiment: Can CQual verify absence of fmt str bugs?
  - Sound whole-program analysis
- Early indications: 1) polymorphic type inf + partial field sensitivity help enormously; 2) FPs are very rare.

	Bugs/Warnings/Manual Annotation?		LOC
Program	Monomorphic	Poly+field sens.	.c / .i
muh	1/12/yes(×6)	1/1/none	3k / 103k
cfengine	1/ 5/yes	1/3/none	24k / 126k
bftpd	1/ 2/yes(×1)	1/1/none	2k / 34k
mars_nwe	0/0/yes(×2)	0/0/none	21k / 73k
sshd	0/0/yes(×12)	0/0/none	26k / 221k
apache	0/0/yes (×2)	0/0/none	33k / 136k
(4 others)	0/0/none	0/0/none	83k / 163k

Work in progress

- Goal: Build a Linux kernel verifiably free of u/k bugs
  - Whole-kernel analysis (5 MLoC), using modular CFL reachability for space efficiency
  - Re-write hard-to-verify code using cleaner idioms
  - Hypothesis: tools can improve software security by gently steering developers toward safer coding styles

- Goal: Verify that Debian is free of format string bugs
  - Whole-program analysis (3000 packages, 50+ MLoC), using modular analysis and parallelization
  - Become part of Debian release/QA process?

Concluding thoughts

- Bugfinding is good. Verification is even better.
- Think big. Experiment bigger.

Questions?