Automated Whitebox Fuzz Testing

Patrice Godefroid (Microsoft Research)
Michael Y. Levin (Microsoft Center for Software Excellence)
David Molnar (UC-Berkeley, done while visiting MSR)
Fuzz Testing

• Send “random” data to application
  – B. Miller et al.; inspired by line noise
• Fuzzing well-formed “seed”
• **Heavily** used in security testing
  – e.g. July 2006 “Month of Browser Bugs”
  – Literally dozens of fuzz testing tools out there
Whitebox Fuzzing

- Combine fuzz testing with **dynamic test generation**
  - Run the code with some initial ‘seed’ input
  - Collect constraints on input with symbolic execution
  - Generate new constraints
  - Solve constraints with constraint solver
  - Synthesize new inputs

- Leverages **Directed Automated Random Testing (DART)**
  ( [Godefroid-Klarlund-Sen-05,...])

- See also **EGT/EXE/KLEE** [Cadar-Engler-05, Cadar-Ganesh-Pawlowski-Engler-Dill-06, Dunbar-Cadar-Pawlowski-Engler-08,...]
void top(char input[4])
{
    int cnt = 0;
    if (input[0] == 'b') cnt++;
    if (input[1] == 'a') cnt++;
    if (input[2] == 'd') cnt++;
    if (input[3] == '!') cnt++;
    if (cnt >= 3) crash();
}
Dynamic Test Generation

```c
void top(char input[4])
{
    int cnt = 0;
    if (input[0] == 'b') cnt++;
    if (input[1] == 'a') cnt++;
    if (input[2] == 'd') cnt++;
    if (input[3] == '!') cnt++;
    if (cnt >= 3) crash();
}
```

input = “good”

Collect constraints from trace
Create new constraints
Solve new constraints → new input.
Depth-First Search

```c
void top(char input[4])
{
    int cnt = 0;
    if (input[0] == 'b') cnt++;
    if (input[1] == 'a') cnt++;
    if (input[2] == 'd') cnt++;
    if (input[3] == '!') cnt++;
    if (cnt >= 3) crash();
}
```
void top(char input[4])
{
    int cnt = 0;
    if (input[0] == 'b') cnt++;
    if (input[1] == 'a') cnt++;
    if (input[2] == 'd') cnt++;
    if (input[3] == '!') cnt++;
    if (cnt >= 3) crash();
}
Depth-First Search

```c
void top(char input[4])
{
    int cnt = 0;
    if (input[0] == 'b') cnt++;
    if (input[1] == 'a') cnt++;
    if (input[2] == 'd') cnt++;
    if (input[3] == '!') cnt++;
    if (cnt >= 3) crash();
}
```
Key Idea: One Trace, Many Tests

Office 2007 application:
Time to **gather constraints**: 25m30s
Tainted branches/trace: ~1000

Time per branch to
**solve**,
generate new test,
check for crashes: ~1s

Therefore, solve+check **all** branches
for each trace!
void top(char input[4])
{
    int cnt = 0;
    if (input[0] == 'b') cnt++;
    if (input[1] == 'a') cnt++;
    if (input[2] == 'd') cnt++;
    if (input[3] == '!') cnt++;
    if (cnt >= 3) crash();
}

"Generation 1" test cases
The Search Space

```c
void top(char input[4])
{
    int cnt = 0;
    if (input[0] == 'b') cnt++;
    if (input[1] == 'a') cnt++;
    if (input[2] == 'd') cnt++;
    if (input[3] == '!') cnt++;
    if (cnt >= 3) crash();
}
```
SAGE Architecture
(Scalable Automated Guided Execution)

Check for Crashes (AppVerifier)

Trace Program (Nirvana)

Gather Constraints (Truscan)

Solve Constraints (Disolver)

Input0

Trace File

Constraints

Input1

Input2

... InputN
Initial Experiences with SAGE

- Since 1st MS internal release in April’07: dozens of new security bugs found (most missed by blackbox fuzzers, static analysis)

- Apps: image processors, media players, file decoders,... Confidential!

- Many bugs found rated as “security critical, severity 1, priority 1”

- Now used by several teams regularly as part of QA process

- Credit is due to the entire SAGE team and users:
  - CSE: Michael Levin (DevLead), Christopher Marsh, Dennis Jeffries (intern’06), Adam Kiezun (intern’07) Plus Nirvana/iDNA/TruScan contributors.
  - MSR: Patrice Godefroid, David Molnar (intern’07) (+ constraint solver Disolver)
  - Plus work of many beta users who found and filed most of these bugs!
ANI Parsing - MS07-017

Initial input

RIFF...ACONLIST
B...INFOINAM....
3D Blue Alternate v1.1..IART....
................
1996..anih$...$.
................
.................
..rate..........
...........seq ..
...............
..LIST....framic
on......... ..

Crashing test case

RIFF...ACONB
B...INFOINAM....
3D Blue Alternate v1.1..IART....
................
1996..anih$...$.
................
.................
..rate..........
...........seq ..
...............
..anih....framic
on......... ..

ANI Parsing - MS07-017

Initial input

Crashing test case

Only 1 in $2^{32}$ chance at random!
Initial Experiments

- #Instructions and Input size largest seen so far

<table>
<thead>
<tr>
<th>App Tested</th>
<th>#Tests</th>
<th>Mean Depth</th>
<th>Mean #Instr.</th>
<th>Mean Size</th>
</tr>
</thead>
<tbody>
<tr>
<td>ANI</td>
<td>11468</td>
<td>178</td>
<td>2,066,087</td>
<td>5,400</td>
</tr>
<tr>
<td>Media 1</td>
<td>6890</td>
<td>73</td>
<td>3,409,376</td>
<td>65,536</td>
</tr>
<tr>
<td>Media 2</td>
<td>1045</td>
<td>1100</td>
<td>271,432,489</td>
<td>27,335</td>
</tr>
<tr>
<td>Media 3</td>
<td>2266</td>
<td>608</td>
<td>54,644,652</td>
<td>30,833</td>
</tr>
<tr>
<td>Media 4</td>
<td>909</td>
<td>883</td>
<td>133,685,240</td>
<td>22,209</td>
</tr>
<tr>
<td>Compression</td>
<td>1527</td>
<td>65</td>
<td>480,435</td>
<td>634</td>
</tr>
<tr>
<td>Office 2007</td>
<td>3008</td>
<td>6502</td>
<td>923,731,248</td>
<td>45,064</td>
</tr>
</tbody>
</table>
Zero to Crash in 10 Generations

• Starting with 100 zero bytes ...

• SAGE generates a crashing test:

```
00000000h: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ; ................
00000010h: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ; ................
00000020h: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ; ................
00000030h: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ; ................
00000040h: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ; ................
00000050h: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ; ................
00000060h: 00 00 00 00 ; ....
```

Generation 0 – initial input – 100 bytes of “00”
Zero to Crash in 10 Generations

• Starting with 100 zero bytes ...

• SAGE generates a crashing test:

```
00000000h:  52 49 46 46 00 00 00 00 00 00 00 00 00 00 00 00 ; RIFF............
00000010h:  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ; ................
00000020h:  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ; ................
00000030h:  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ; ................
00000040h:  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ; ................
00000050h:  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ; ................
00000060h:  00 00 00 00
```

Generation 1
Zero to Crash in 10 Generations

- Starting with 100 zero bytes ... 

- SAGE generates a crashing test:

<table>
<thead>
<tr>
<th>Address</th>
<th>Hexadecimal</th>
<th>ASCII</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>00000000h</td>
<td>52 49 46 46 00 00 00 00 ** ** ** 20 00 00 00 00</td>
<td>RIFF...***...</td>
<td>RIFF header with null bytes and garbage</td>
</tr>
<tr>
<td>00000010h</td>
<td>00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00</td>
<td></td>
<td>Uses the RIFF format with null bytes</td>
</tr>
<tr>
<td>00000020h</td>
<td>00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00</td>
<td></td>
<td>Alters the RIFF header</td>
</tr>
<tr>
<td>00000030h</td>
<td>00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00</td>
<td></td>
<td>Continues altering the RIFF header</td>
</tr>
<tr>
<td>00000040h</td>
<td>00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00</td>
<td></td>
<td>Further modifications of the RIFF header</td>
</tr>
<tr>
<td>00000050h</td>
<td>00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00</td>
<td></td>
<td>Keeps altering the RIFF header</td>
</tr>
<tr>
<td>00000060h</td>
<td>00 00 00 00</td>
<td></td>
<td>Continues to modify the RIFF header</td>
</tr>
</tbody>
</table>

Generation 2
Zero to Crash in 10 Generations

• Starting with 100 zero bytes ...

• SAGE generates a crashing test:

```
00000000h: 52 49 46 46 3D 00 00 00 ** ** ** 20 00 00 00 00 ; RIFF=....
00000010h: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ; ................
00000020h: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ; ................
00000030h: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ; ................
00000040h: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ; ................
00000050h: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ; ................
00000060h: 00 00 00 00
```

Generation 3
Zero to Crash in 10 Generations

• Starting with 100 zero bytes ...

• SAGE generates a crashing test:

```
00000000h: 52 49 46 46 3D 00 00 00 ** ** ** 20 00 00 00 00 ; RIFF=...*** ....
00000010h: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ; ................
00000020h: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ; ................
00000030h: 00 00 00 00 73 74 72 68 00 00 00 00 00 00 00 00 ; ....strh........
00000040h: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ; ................
00000050h: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ; ................
00000060h: 00 00 00 00
```

Generation 4
Zero to Crash in 10 Generations

• Starting with 100 zero bytes ...

• SAGE generates a crashing test:

```
00000000h: 52 49 46 46 3D 00 00 00 ** ** ** 20 00 00 00 00 ; RIFF=...*** ..... 
00000010h: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ; .................. 
00000020h: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ; .................. 
00000030h: 00 00 00 00 73 74 72 68 00 00 00 00 76 69 64 73 ; .....strh...vids 
00000040h: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ; .................. 
00000050h: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ; .................. 
00000060h: 00 00 00 00 ; ....
```

Generation 5
Zero to Crash in 10 Generations

- Starting with 100 zero bytes ...

- SAGE generates a crashing test:

<table>
<thead>
<tr>
<th>Memory Address</th>
<th>Hex Content</th>
</tr>
</thead>
<tbody>
<tr>
<td>00000000h</td>
<td>52 49 46 46 3D 00 00 00 ** ** ** 20 00 00 00 00 ; RIFF=...*** ....</td>
</tr>
<tr>
<td>00000010h</td>
<td>00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ; ................</td>
</tr>
<tr>
<td>00000020h</td>
<td>00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ; ................</td>
</tr>
<tr>
<td>00000030h</td>
<td>00 00 00 00 73 74 72 68 00 00 00 00 76 69 64 73 ; ....strh....vids</td>
</tr>
<tr>
<td>00000040h</td>
<td>00 00 00 00 73 74 72 66 00 00 00 00 00 00 00 00 ; ....strf.........</td>
</tr>
<tr>
<td>00000050h</td>
<td>00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ; .................</td>
</tr>
<tr>
<td>00000060h</td>
<td>00 00 00 00</td>
</tr>
</tbody>
</table>

Generation 6
Zero to Crash in 10 Generations

• Starting with 100 zero bytes ...

• SAGE generates a crashing test:

```
00000000h: 52 49 46 46 3D 00 00 00 ** ** ** 20 00 00 00 00 ; RIFF=...*** ..... 
00000010h: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ; ................
00000020h: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ; ................
00000030h: 00 00 00 00 73 74 72 68 00 00 00 00 76 69 64 73 ; ....strh....vids
00000040h: 00 00 00 00 73 74 72 66 00 00 00 00 28 00 00 00 ; ....strf....{...
00000050h: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ; ................
00000060h: 00 00 00 00
```

Generation 7
Zero to Crash in 10 Generations

• Starting with 100 zero bytes ...

• SAGE generates a crashing test:

<table>
<thead>
<tr>
<th>Offset</th>
<th>Hex</th>
<th>ASCII</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>00000000h:</td>
<td>52 49 46 46 3D 00 00 00</td>
<td>** ** ** 20 00 00 00 00</td>
<td>RIFF=***....</td>
</tr>
<tr>
<td>00000010h:</td>
<td>00 00 00 00 00 00 00 00</td>
<td></td>
<td></td>
</tr>
<tr>
<td>00000020h:</td>
<td>00 00 00 00 00 00 00 00</td>
<td></td>
<td></td>
</tr>
<tr>
<td>00000030h:</td>
<td>00 00 00 00 73 74 72 68</td>
<td>73 74 72 68 00 00 00 00</td>
<td>strh....vids</td>
</tr>
<tr>
<td>00000040h:</td>
<td>00 00 00 00 73 74 72 66</td>
<td>73 74 72 66 00 00 00 00</td>
<td>strf....(</td>
</tr>
<tr>
<td>00000050h:</td>
<td>00 00 00 00 00 00 00 00</td>
<td>00 00 00 00 C9 9D E4 4E</td>
<td></td>
</tr>
<tr>
<td>00000060h:</td>
<td>00 00 00 00</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Generation 8
Zero to Crash in 10 Generations

• Starting with 100 zero bytes ...

• SAGE generates a crashing test:

```
00000000h:  52 49 46 46 3D 00 00 00 ** ** ** 20 00 00 00 00 ; RIFF=...*** ....
00000010h:  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ; ................
00000020h:  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ; ................
00000030h:  00 00 00 00 73 74 72 68 00 00 00 00 76 69 64 73 ; ....strh....vids
00000040h:  00 00 00 00 73 74 72 66 00 00 00 00 28 00 00 00 ; ....strf....(...
00000050h:  00 00 00 00 00 00 00 00 00 00 00 00 00 00 01 00 00 00 ; ................
00000060h:  00 00 00 00
```

Generation 9
Zero to Crash in 10 Generations

• Starting with 100 zero bytes ...

• SAGE generates a crashing test:

<table>
<thead>
<tr>
<th>Offset</th>
<th>Hex</th>
<th>Binary</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>00000000h</td>
<td>52 49 46 46 3D 00 00 00 ** ** ** 20 00 00 00 00</td>
<td></td>
<td>RIFF=...*** .....</td>
</tr>
<tr>
<td>00000010h</td>
<td>00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00</td>
<td></td>
<td>................</td>
</tr>
<tr>
<td>00000020h</td>
<td>00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00</td>
<td></td>
<td>................</td>
</tr>
<tr>
<td>00000030h</td>
<td>00 00 00 00 73 74 72 68 00 00 00 00 76 69 64 73</td>
<td></td>
<td>....strh....vids</td>
</tr>
<tr>
<td>00000040h</td>
<td>00 00 00 00 73 74 72 66 B2 75 76 3A 28 00 00 00</td>
<td></td>
<td>....strfuv:...</td>
</tr>
<tr>
<td>00000050h</td>
<td>00 00 00 00 00 00 00 00 00 00 00 00 00 01 00 00 00</td>
<td></td>
<td>................</td>
</tr>
<tr>
<td>00000060h</td>
<td>00 00 00 00</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Generation 10 – bug ID 1212954973!

Found after only 3 generations starting from “well-formed” seed file
Different Crashes From Different Initial Input Files

<table>
<thead>
<tr>
<th>Bug ID</th>
<th>seed1</th>
<th>seed2</th>
<th>seed3</th>
<th>seed4</th>
<th>seed5</th>
<th>seed6</th>
<th>seed7</th>
<th>100 zero bytes</th>
</tr>
</thead>
<tbody>
<tr>
<td>1867196225</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td></td>
<td>X</td>
<td>X</td>
<td>X</td>
<td></td>
</tr>
<tr>
<td>2031962117</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td></td>
<td>X</td>
<td>X</td>
<td>X</td>
<td></td>
</tr>
<tr>
<td>612334691</td>
<td></td>
<td>X</td>
<td>X</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1061959981</td>
<td></td>
<td>X</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>1212954973</strong></td>
<td></td>
<td></td>
<td>X</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>X</td>
</tr>
<tr>
<td>1011628381</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>X</td>
<td></td>
<td></td>
<td>X</td>
</tr>
<tr>
<td>842674295</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>X</td>
<td></td>
<td></td>
</tr>
<tr>
<td>1246509355</td>
<td></td>
<td>X</td>
<td></td>
<td></td>
<td></td>
<td>X</td>
<td></td>
<td>X</td>
</tr>
<tr>
<td>1527393075</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>X</td>
<td></td>
</tr>
<tr>
<td>1277839407</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>X</td>
<td></td>
</tr>
<tr>
<td>1951025690</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>X</td>
<td></td>
</tr>
</tbody>
</table>

**Media 1:** 60 machine-hours, 44,598 total tests, 357 crashes, 12 bugs
Most Bugs Found are “Shallow”
Most Bugs Found are “Shallow”
Most Bugs Found are “Shallow”

Tried two different rules for picking next child to expand: random and “coverage-maximizing.” Both rules had same result: most bugs found in early generations.

Why should this be so?
Code Coverage and New Bugs

New bucket found for the first time
Code Coverage and New Bugs

Does coverage correlate with new bugs or not? Data contradictory.
More Observations

• Percentage of “tainted” instructions small
• “Related Constraint Optimization” big win
• Almost every constraint solved in < 1s each
  – Great! but 1000s of them per trace...
  – Why so fast?
    • SAGE less precise handling memory than EXE/KLEE
      – Concretize all pointers
    • Constraint subsumption + simplification on the fly
    • Disolver is fast (but low-level; integer inequalities only)
More Observations

• Binaries + scale = adoption win
  – “Feels like” black box fuzzer to testers
  – “If you can run it, you can test it with SAGE”
  – Most bugs filed after internal SAGE release

• Binary analysis also catches compiler quirks
  – Example: GCC 4.1 removes “assert(a + 100 > a)”

• Main downsides
  – Less info about code (partially address w/symbols)
  – Scale (millions of instructions per trace)
Catchconv (Work-in-Progress)

- D. Molnar and D. Wagner, August 2006-now
- Uses Valgrind dynamic binary instrumentation
- Current focus: Linux media players
  - Four bugs to mplayer developers (3 fixed, 1 “hard”)
  - Largest target so far: Flash Player 9/example.swf
    - ~200MM instructions, 438K STP constraints
    - Issue: some Flash test cases crash only under Valgrind!
- Code repository on Sourceforge
  - http://sourceforge.net/cvs/?group_id=187658
0x8048102:  popl %esi

------- IMark(0x8048102, 1) -------
PUT(60) = 0x8048102:I32
t4 = GET:I32(16)
t3 = LDle:I32(t4)
PUT(16) = Add32(t4,0x4:I32)
PUT(24) = t3
0x8048102: popl %esi

----- IMark(0x8048102, 1) ----- 
PUT(60) = 0x8048102:I32 
t4 = GET:I32(16) 
t3 = LDle:I32(t4) 
t25 = Add32(t4,0x4:I32) 
PUT(24) = t3

MemState0p22034th1 : ARRAY BITVECTOR(32) OF BITVECTOR(32); 
MaSt0p22034th1 : ARRAY BITVECTOR(12) OF BITVECTOR(32); 
[...]
MaSt2p22034th1 : ARRAY BITVECTOR(12) OF BITVECTOR(32) = 
MaSt1p22034th1 WITH [0hex03C] := 0hex08048102;

ASSERT(CVOe1t4p22034th1 = MaSt2p22034th1[0hex010]);
ASSERT(CVOe1t3p22034th1 = MemState0p22034th1[CVOe1t4p22034th1]);
ASSERT(CVOe1t25p22034th1 = BVPLUS(32,CVOe1t4p22034th1,0hex00000004));

MaSt3p22034th1 : ARRAY BITVECTOR(12) OF BITVECTOR(32) = 
MaSt2p22034th1 WITH [0hex018] := CVOe1t3p22034th1;
Metafuzz (Work-In-Progress)

• Empirical questions
  – Fast black-box or slower white-box tester?
  – Where are most bugs found in the search?
  – Does code coverage correlate with bugs found?

• Metafuzz: reporting infrastructure
  – Test case info goes to DB in the sky
    • #blocks covered, #new blocks, bug found?, seed file, etc.
  – Currently: zzuf and catchconv feeders to Metafuzz
  – Plan: charts, data export, bugzilla integration
Blackbox vs. Whitebox Fuzzing

• Cost/precision tradeoff
  – Blackbox is lightweight, easy and fast, but poor code coverage
  – Whitebox is smarter, but complex and slower
  – Recent “semi-whitebox” approaches
    • Less smart but more lightweight: Flayer (taint-flow analysis, may generate false alarms), Bunny-the-fuzzer (taint-flow, source-based, heuristics to fuzz based on input usage), autodafe, etc.

• Which is more effective at finding bugs? It depends…
  – Many apps are so buggy, any form of fuzzing finds bugs!
  – Once low-hanging bugs are gone, fuzzing must become smarter: use whitebox and/or user-provided guidance (grammars, etc.)

• Bottom-line: in practice, use both!
Summary

• Symbolic execution **scales**
  – SAGE most successful “DART implementation”
  – Dozens of serious bugs, used regularly at MSFT

• Existing test suites become security tests
  – Integrate with regression test tools?

• Open empirical questions

• Future of fuzz testing?
Thank you!

Questions?

dmolnar@eecs.berkeley.edu