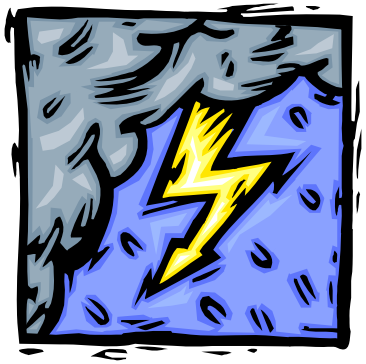
Automated Whitebox Fuzz Testing

Patrice Godefroid (Microsoft Research)
Michael Y. Levin (Microsoft Center for Software Excellence)
David Molnar (UC-Berkeley & 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”



Whitebox Fuzzing

- Combine fuzz testing with **dynamic test generation**
  - **Run the code** with its input
  - **Collect constraints on inputs** with symbolic execution
  - **Generate new constraints**
  - **Solve constraints** with constraint solver
  - **Synthesize new inputs**
  - Leverages **Directed Automated Random Testing** (**DART**) ([Godefroid-Klarlund-Sen PLDI 2005,…])

Dynamic Test Generation

input = “good”

```
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

```
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”

I_0 != ‘b’

I_1 != ‘a’

I_2 != ‘d’

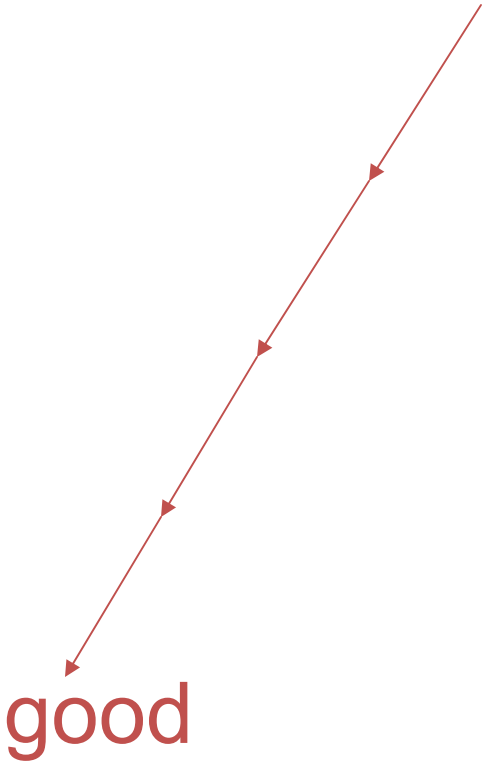
I_3 != ‘!’

Collect constraints from trace

Create new constraints

Solve new constraints → new input.

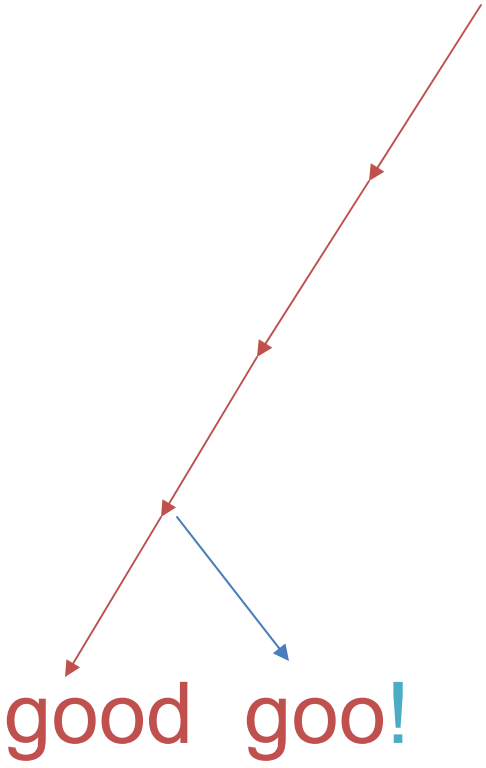
Depth-First Search




```
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();
}
```

I_0 != 'b'
I_1 != 'a'
I_2 != 'd'
I_3 != '!'

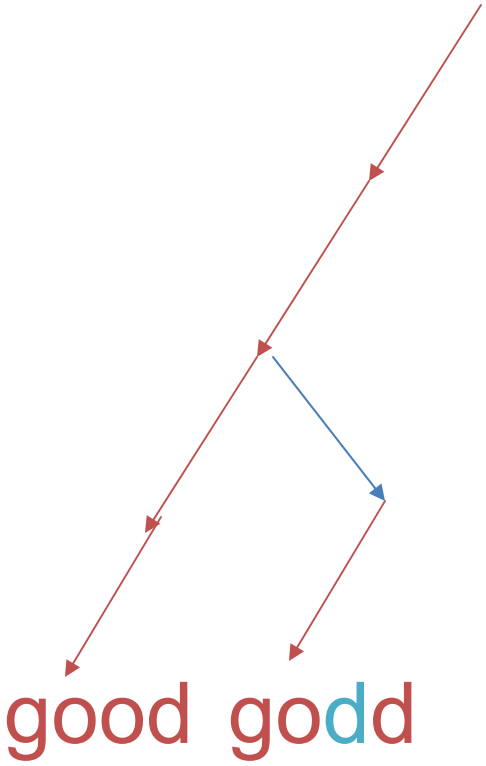
Depth-First Search




```
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();
}
```

I_0 != 'b'
I_1 != 'a'
I_2 != 'd'
I_3 == '!'

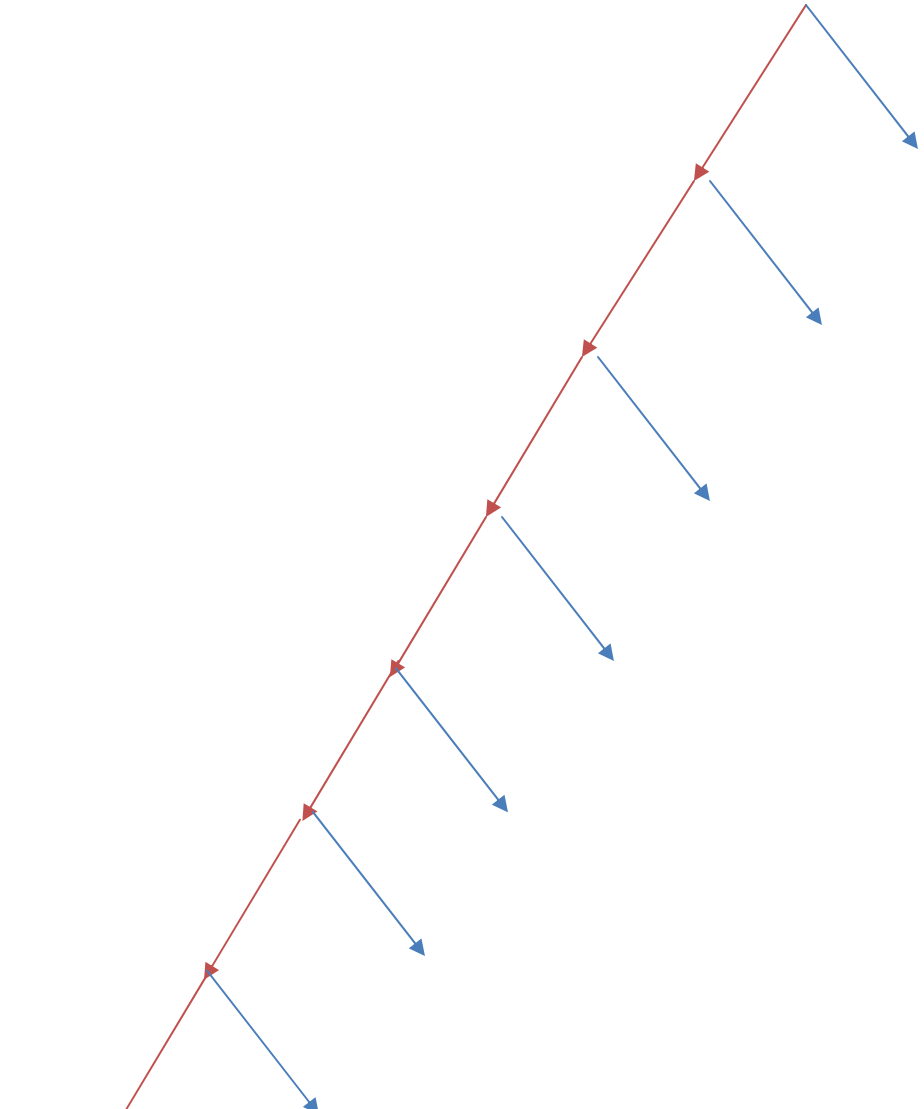
Depth-First Search




```
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();
}
```

I_0 != ‘b’
I_1 != ‘a’
I_2 == ‘d’
I_3 != ‘!’

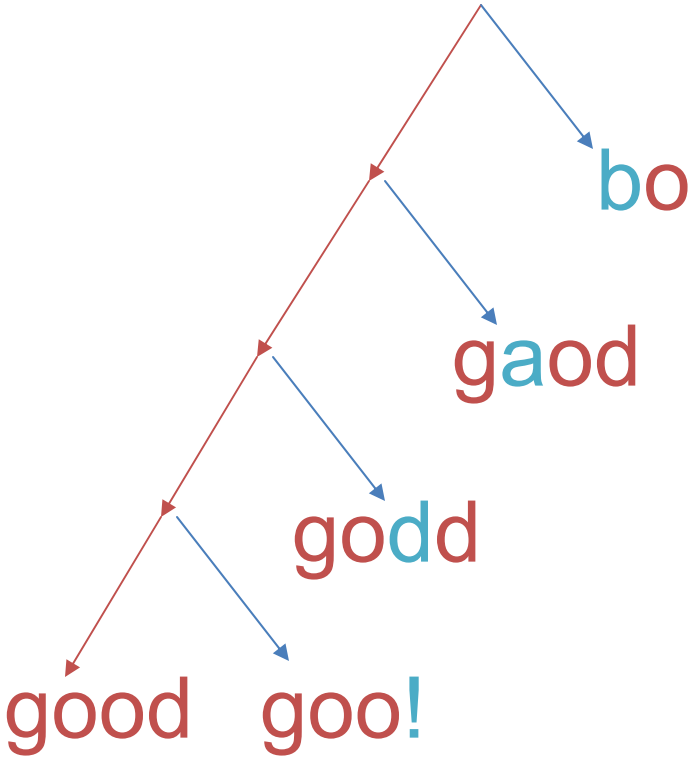
Key Idea: One Trace, Many Tests



Office 2007 application:
Time to **gather constraints**: 25m30s
Tainted branches/trace: ~1000
Time/branch to **solve**,
generate new test,
check for crashes: ~1s

Therefore, solve+check **all** branches for each trace!

Generational Search



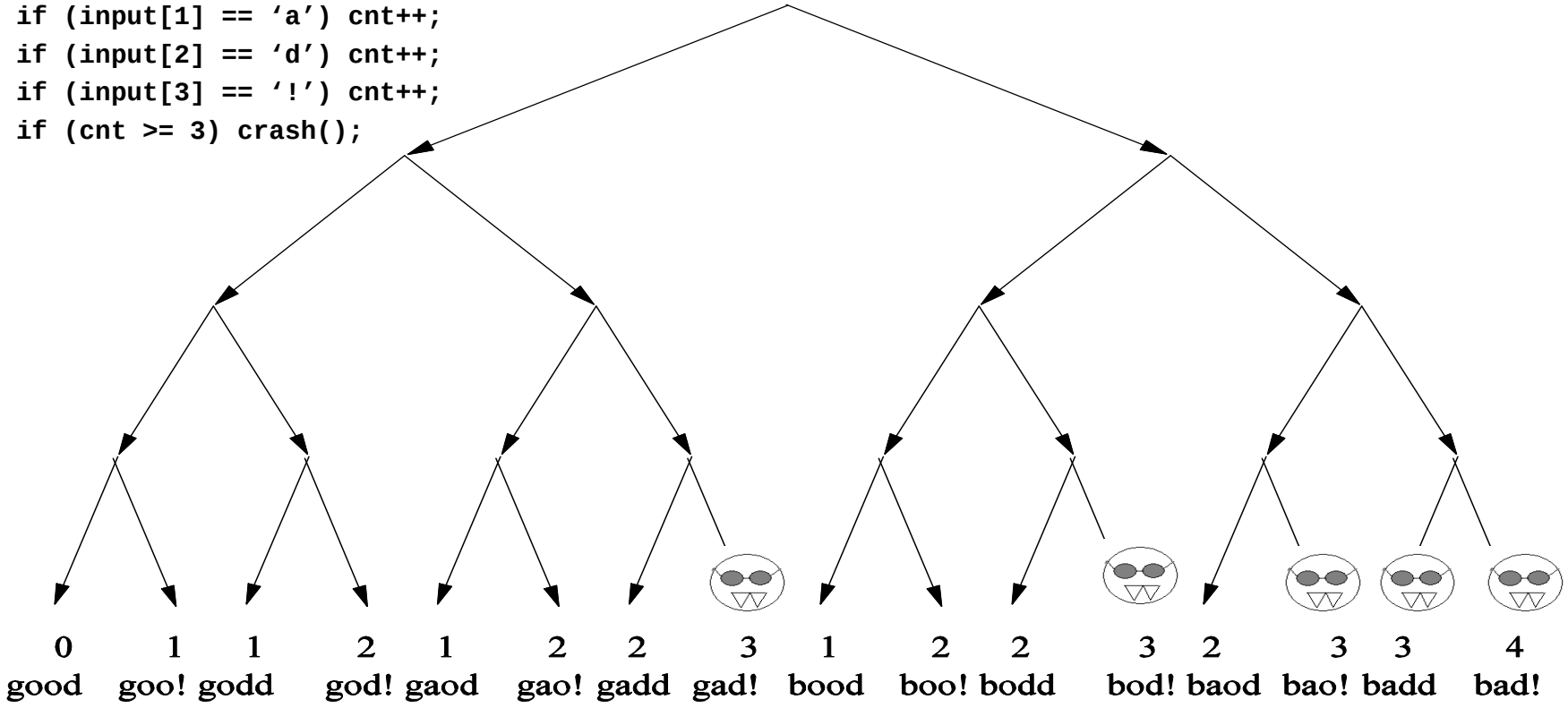

```
void top(char input[4])
{
    int cnt = 0;
    if (input[0] == 'b') cnt++;
    if (input[1] == 'a') cnt++;
    if (input[2] == 'd') cnt++;
    if (input[3] == '!') cnt++;
    if (cnt >= 3) crash();
}
```

I_0 == 'b'
I_1 == 'a'
I_2 == 'd'
I_3 == '!'

"Generation 1" test cases

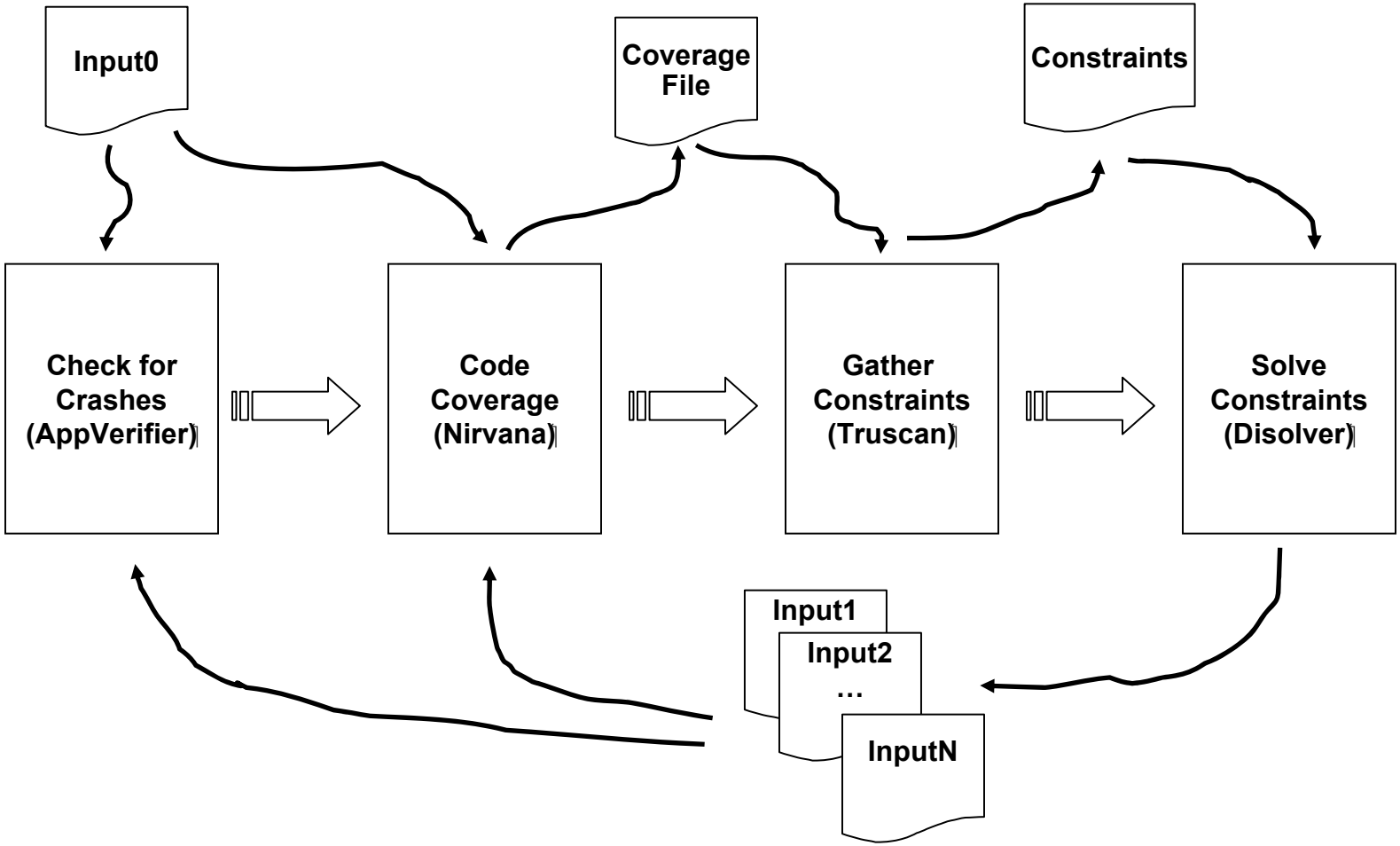
The Search Space

```
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)



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 test teams across Microsoft

- 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

Critical, **out-of-band** security patch; affected Vista

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

Seed file

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

SAGE-generated crashing test case

ANI Parsing - MS07-017

Critical, **out-of-band** security patch; affected Vista

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

Seed file

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

SAGE-generated
crashing test case

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

Initial Experiments

- #Instructions and Input size largest seen so far

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

Zero to Crash in 10 Generations

- Starting with 100 zero bytes …
- SAGE generates a crashing test for Media1 parser:

```
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 – seed file

Zero to Crash in 10 Generations

- Starting with 100 zero bytes …
- SAGE generates a crashing test for Media1 parser:

```
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 for Media1 parser:

```
00000000h: 52 49 46 46 00 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 2

Zero to Crash in 10 Generations

- Starting with 100 zero bytes …
- SAGE generates a crashing test for Media1 parser:

```
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 for Media1 parser:

```
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 for Media1 parser:

```
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 for Media1 parser:

```
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 00 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 6

Zero to Crash in 10 Generations

- Starting with 100 zero bytes …
- SAGE generates a crashing test for Media1 parser:

```
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 for Media1 parser:

```
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 C9 9D E4 4E ; ............É□äN
00000060h: 00 00 00 00                                     ; ....
```

Generation 8

Zero to Crash in 10 Generations

- Starting with 100 zero bytes …
- SAGE generates a crashing test for Media1 parser:

```
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 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 for Media1 parser:

```
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 B2 75 76 3A 28 00 00 00 ; ....strf²uv:(...
00000050h: 00 00 00 00 00 00 00 00 00 00 00 00 01 00 00 00 ; ................
00000060h: 00 00 00 00                                     ; ....
```

Generation 10 – bug ID 1212954973!

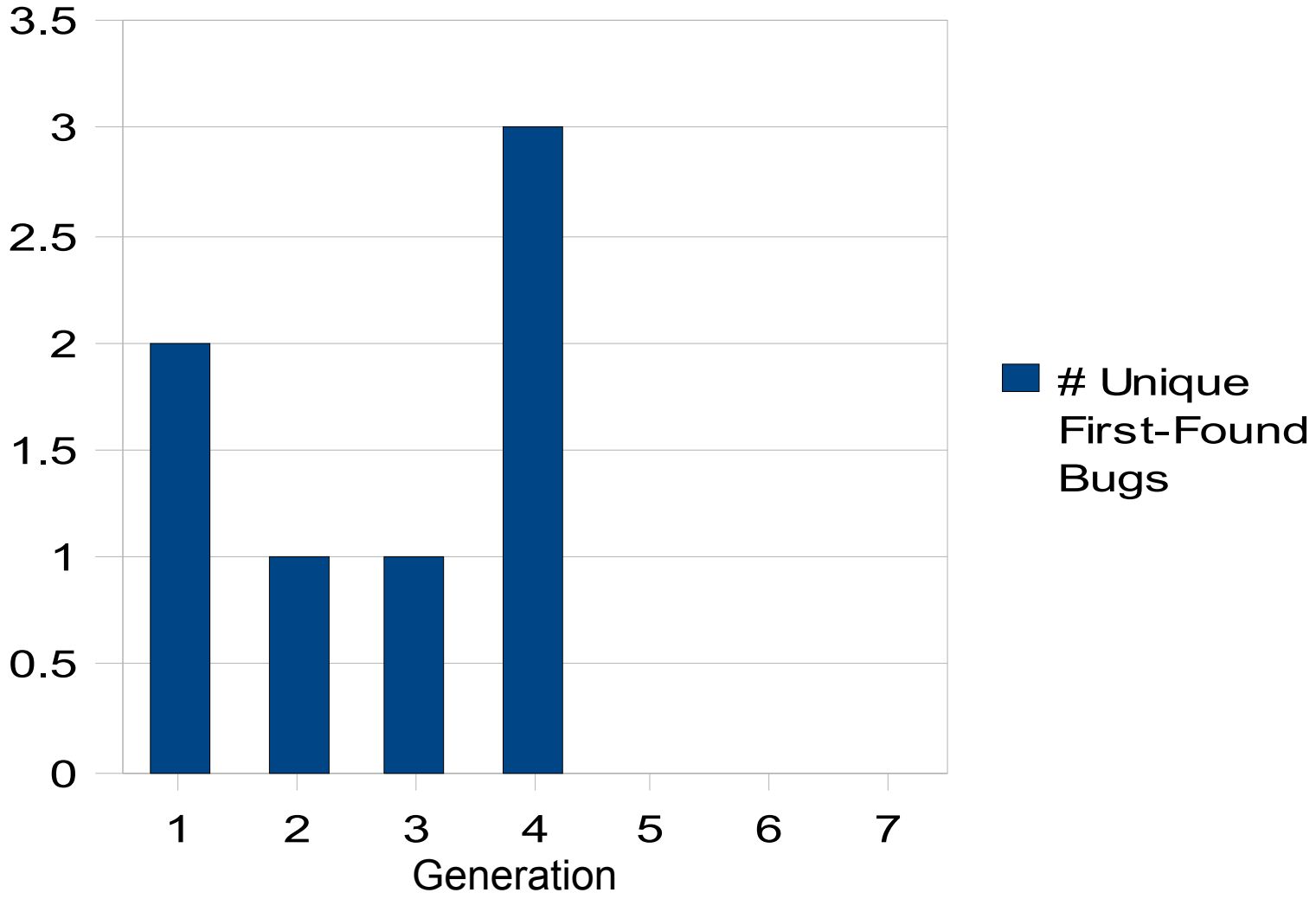
Found after only 3 generations starting from well-formed seed file

Different Seeds, Different Crashes

Bug ID	seed1	seed2	seed3	seed4	seed5	seed6	seed7	100 zero bytes
186719622	X	X	X		X		X	
203196211	X	X	X		X		X	
612334691		X	X					
106195998			X		X			
121295497			X					X
101162838			X		X			X
842674295					X			
124650935			X		X			X
152739307							X	
127783940							X	
195102569			X					

Media 1: 60 machine-hours, 44598 total tests, 357 crashes, 12 bugs

Most Bugs Found are “Shallow”



Blackbox vs. Whitebox Fuzzing

- Different cost/precision tradeoffs
  - Blackbox is lightweight, easy and fast, but poor coverage
  - Whitebox is smarter, but complex and slower
  - Note: 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**!

Related Work

- Dynamic test generation (Korel, Gupta-Mathur-Soffa, etc.)
  - Target specific statement; DART tries to cover "most" code
- Static Test Generation: hard when symbolic execution imprecise
- Other "DART implementations":
  - **EXE/EGT** (Stanford): independent ['05-'06] closely related work
  - **CUTE/jCUTE** (UIUC/Berkeley): same as Bell Labs DART implementation
  - **PEX** (MSR) implements DART for .NET binaries in conjunction with "parameterized-unit tests" for unit testing of .NET programs
  - **YOGI** (MSR) implements DART to check the feasibility of program paths generated statically using a SLAM-like tool
  - **Vigilante** (MSR) implements DART to generate worm filters
  - **BitScope** (CMU/Berkeley) implements DART for malware analysis
  - **Catchconv** (Berkeley) extends DART to check signed/unsigned integer errors
  - More..

SAGE Summary

- Symbolic execution **scales**
  - SAGE most successful "DART implementation"
  - Dozens of serious bugs, used daily at MSFT
- Existing test suites become security tests
- What makes it so effective?
  - Works on large applications (not unit test)
  - Fully automated (focus on file/network fuzzing)
  - Easy to deploy (dynamic binary instrumentation – any lang. or build process!)
- Future of fuzz testing?

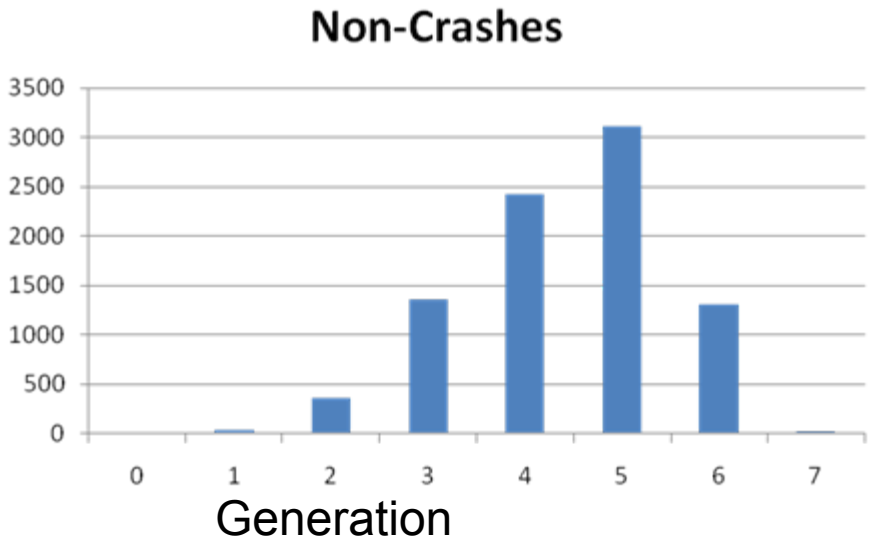
Thank you!

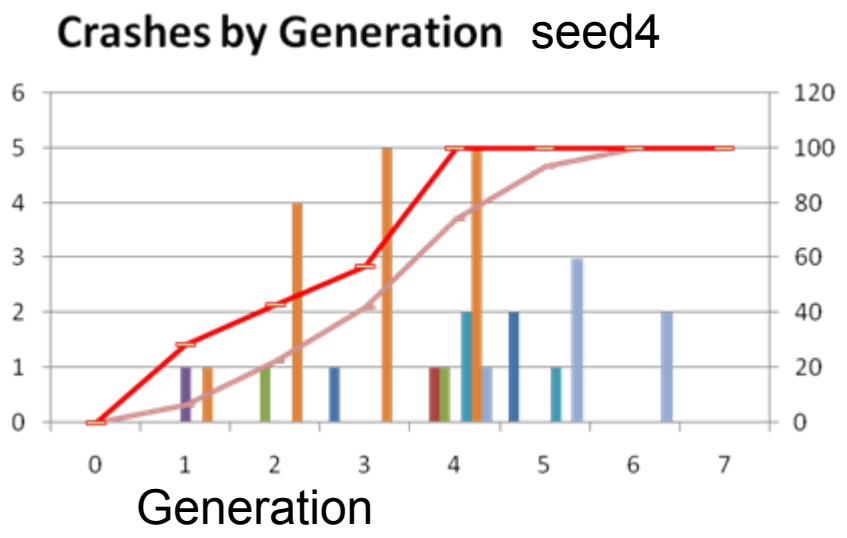
Questions?

dmolnar@eecs.berkeley.edu

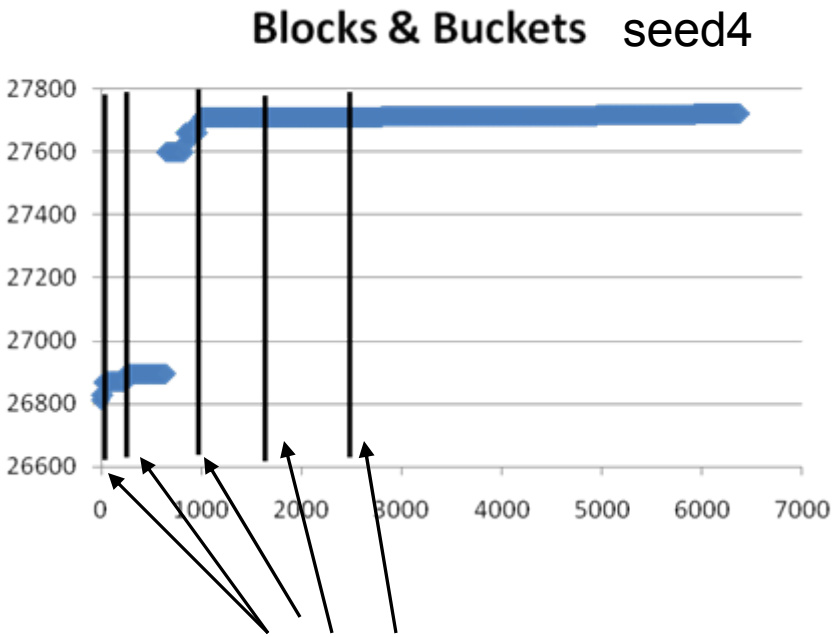
Backup Slides

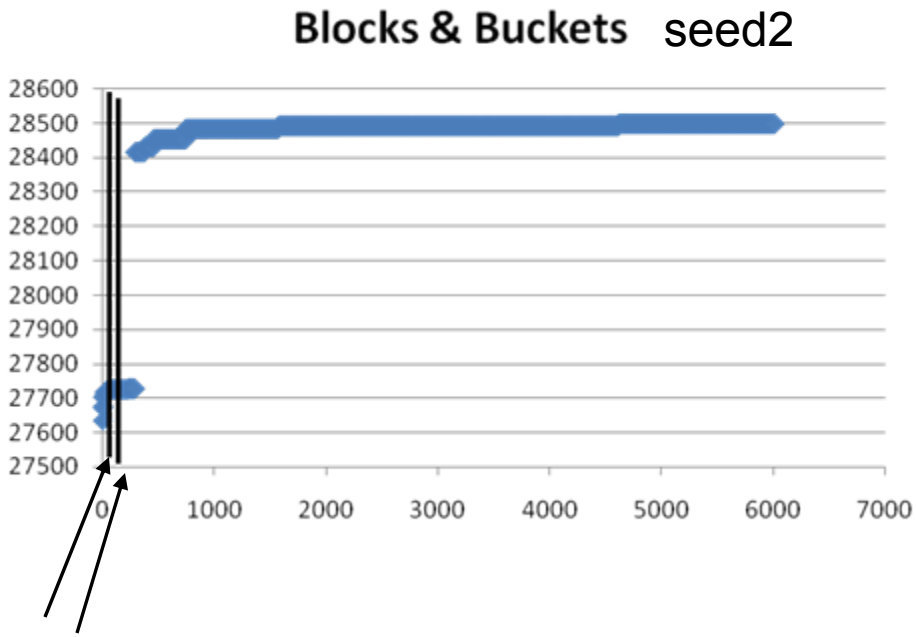
Most Bugs Found are “Shallow”





Coverage and New crashes: Low Correlation





New bucket found for the first time