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,...])
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"
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”

I₀ != ‘b’
I₁ != ‘a’
I₂ != ‘d’
I₃ != ‘!’

Collect constraints from trace
Create new constraints
Solve new constraints \(\rightarrow\) new input.
void top(char input[4])
{
    int cnt = 0;
    if (input[0] == 'b') cnt++; I_0 != 'b'
    if (input[1] == 'a') cnt++; I_1 != 'a'
    if (input[2] == 'd') cnt++; I_2 != 'd'
    if (input[3] == '!') cnt++; I_3 != '!' 
    if (cnt >= 3) crash();
}

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();
}

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/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
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)

Code Coverage (Nirvana)

Gather Constraints (Truscan)

Solve Constraints (Disolver)

Coverage File

Constraints

Input0

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 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

Seed file

SAGE-generated crashing test case
ANI Parsing - MS07-017

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

Seed file

SAGE-generated 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 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:

<table>
<thead>
<tr>
<th>Generation 2</th>
</tr>
</thead>
<tbody>
<tr>
<td>00000000h: 52 49 46 46 00 00 00 00 ** ** ** 20 00 00 00 00 ; RIFF....***....</td>
</tr>
<tr>
<td>00000010h: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ; ................</td>
</tr>
<tr>
<td>00000020h: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ; ................</td>
</tr>
<tr>
<td>00000030h: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ; ................</td>
</tr>
<tr>
<td>00000040h: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ; ................</td>
</tr>
<tr>
<td>00000050h: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ; ................</td>
</tr>
<tr>
<td>00000060h: 00 00 00 00</td>
</tr>
</tbody>
</table>
Zero to Crash in 10 Generations

• Starting with 100 zero bytes ...

• SAGE generates a crashing test for Media1 parser:

<table>
<thead>
<tr>
<th>Generation 3</th>
</tr>
</thead>
<tbody>
<tr>
<td>000000000h: 52 49 46 46 3D 00 00 00 ** ** ** 20 00 00 00 00 ; RIFF=...*** ....</td>
</tr>
<tr>
<td>00000010h: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ; ................</td>
</tr>
<tr>
<td>00000020h: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ; ................</td>
</tr>
<tr>
<td>00000030h: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ; ................</td>
</tr>
<tr>
<td>00000040h: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ; ................</td>
</tr>
<tr>
<td>00000050h: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ; ................</td>
</tr>
<tr>
<td>00000060h: 00 00 00 00</td>
</tr>
</tbody>
</table>
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:

```
000000000h: 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:

<table>
<thead>
<tr>
<th>Address</th>
<th>Hex Contents</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 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:

```plaintext
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 C9 9D E4 4E ; ................. aN
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 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 ; ....strfu:v:....
00000050h:  00 00 00 00 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

<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>186719622</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td></td>
<td>X</td>
<td></td>
<td>X</td>
<td></td>
</tr>
<tr>
<td>203196211</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td></td>
<td>X</td>
<td></td>
<td>X</td>
<td></td>
</tr>
<tr>
<td>612334691</td>
<td>X</td>
<td></td>
<td>X</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>106195998</td>
<td></td>
<td></td>
<td>X</td>
<td></td>
<td>X</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>121295497</td>
<td></td>
<td>X</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>X</td>
<td></td>
</tr>
<tr>
<td>101162838</td>
<td></td>
<td>X</td>
<td></td>
<td></td>
<td>X</td>
<td></td>
<td>X</td>
<td></td>
</tr>
<tr>
<td>842674295</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>X</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>124650935</td>
<td></td>
<td>X</td>
<td></td>
<td></td>
<td>X</td>
<td></td>
<td>X</td>
<td></td>
</tr>
<tr>
<td>552739307</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>X</td>
</tr>
<tr>
<td>527783940</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>X</td>
</tr>
<tr>
<td>195102569</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, 44598 total tests, 357 crashes, 12 bugs
Most Bugs Found are “Shallow”

![Bar chart showing the number of unique first-found bugs per generation. The chart indicates a peak in generation 4 with a significant number of bugs compared to other generations.](chart.png)
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