Theory Plus Practice in Computer Security: Radio Frequency Identification and Whitebox Fuzzing

David Molnar
dmolnar@eecs.berkeley.edu
Radio Frequency Identification
-or-
A Small Scale Problem
With Big Implications
RFID

- RFID = Radio Frequency IDentification
- “tag” : tiny device with antenna, carries information
- Often “passive” : reader signal powers tag
- Can tag per pallet, per item, or per person
RFID Applications

They're Everywhere!
Security and Privacy?

Nanette Miranda, ABC KGO-TV 7 News, 2006
http://www.youtube.com/watch?v=4jpRFgDPWVA
RFID – A Range of Technologies

Computation

e-passport, transit card

library books, car keys

WalMart, Sutter student ID enhanced driver license

10cm 1/2 m 3 m

Intended Read Range
Deployment: Library RFID

Key Problems:
1. If title and author present, learn what someone reads.
3. Many deployed tags have unique IDs “baked in” to low-level collision avoidance protocols.
Deployment: e-Passports

Proposed deployment had **no encryption or authentication**.

Key Problems:
1. Skimming gives information to anyone with a reader.
2. Skimming allows creation of fake passport.
3. Photograph used for biometric authentication; digital photo aids in spoofing face recognition.
After our report, 2,400 negative public comments, and public outcry, U.S. State Department adds encryption to e-passports. Reader obtains encryption key by optically scanning the inside cover of the e-passport.

Addresses security and privacy problems in e-passports. Not generally applicable for other RFID deployments.
Key Problem: Private Authentication
Key Problem: Private Authentication
A Solution With A Scaling Problem

1. **Authentication.** Cannot predict $F(N, N')$.
2. **Privacy.** Cannot distinguish two tag responses.
3. **Scaling.** Alice needs to try one key per tag, if $T$ tags then scales as $O(T)$. Can we do better?

Here $F$ is your favorite pseudo-random function.
A Flawed Attempt At Scaling

1. **Authentication.** Cannot predict $F(\hat{\Phi}, N, N')$.
2. **Scaling.** Alice looks up key corresponding to ID, $O(1)$.
3. **Privacy.** None! Tag sends ID in the clear.
Scaling : A Key Tree
Scaling: A Key Tree

or?

or?
Scaling: A Key Tree

or

or

?
Scaling : A Key Tree

or

or

?
1. **Authentication.** Can’t predict $F(\text{key}, \text{N}, \text{N'})$ in each level.
2. **Privacy.** Cannot distinguish tags without keys.
3. **Scaling.** As $O(\log T)$, can change branching factor.
# Tree Scheme Scales Best

**Asymptotic Scaling:**

<table>
<thead>
<tr>
<th>Quantity</th>
<th>Tree Scheme</th>
<th>Naive</th>
<th>Precomputation</th>
</tr>
</thead>
<tbody>
<tr>
<td>Reader Time</td>
<td>$O(\log T)$</td>
<td>$O(T)$</td>
<td>$O(T^{2/3})$</td>
</tr>
<tr>
<td>Reader Space</td>
<td>$O(1)$</td>
<td>$O(1)$</td>
<td>$O(T^{2/3})$</td>
</tr>
<tr>
<td>Tag Time</td>
<td>$O(\log T)$</td>
<td>$O(1)$</td>
<td>$O(1)$</td>
</tr>
<tr>
<td>Tag Storage</td>
<td>$O(\log T)$</td>
<td>$O(1)$</td>
<td>$O(1)$</td>
</tr>
<tr>
<td>Communication</td>
<td>$O(\log T)$</td>
<td>$O(\log T)$</td>
<td>$O(\log T)$</td>
</tr>
</tbody>
</table>

**Concrete numbers** for our tree scheme + optimizations in paper:

<table>
<thead>
<tr>
<th>Quantity</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Number of Tags (T)</td>
<td>$2^{20} = 1,048,576$</td>
</tr>
<tr>
<td>Reader Time</td>
<td>2048 evaluations of F</td>
</tr>
<tr>
<td>Reader Space</td>
<td>128 bits</td>
</tr>
<tr>
<td>Tag Time</td>
<td>4 evaluations of F</td>
</tr>
<tr>
<td>Tag Space</td>
<td>192 bits</td>
</tr>
<tr>
<td>Communication</td>
<td>168 bits</td>
</tr>
</tbody>
</table>
Theory Plus Practice

Security Analysis of Practical Deployments → Fundamental Problem of Scaling Private Authentication → Novel Algorithm for Solution
Software Security
-or-
Looking For High Value Bugs
Security Bugs Common and Costly

- 6,515 vulnerabilities reported to CERT in 2007
  - Major vendors: Adobe, Apple, Microsoft …
  - Plus many more
    - web.nvd.nist.gov/view/vuln/statistics
- Writing security patch, QA’ing, releasing costly
The "Bug Cycle"
The “Bug Cycle”
Technique: Fuzz Testing

Miller, Fredriksen, and So,
“An Empirical Study of the Reliability of UNIX Utilities”
http://pages.cs.wisc.edu/~bart/fuzz/fuzz.html
Example of Blackbox Fuzz Testing

Now is the time for all good men to come to the aid of their country.

“Seed file”

No A D4 t#e ti2e foZ #lA HoxZ Ae* Lo ZoBe tA %h& a*( oA tXeLr BoAn$Rq!

Program

Crash?
Fuzz Testing Finds Lots of Bugs!

Operating system facilities, such as the kernel and utility programs, are typically assumed to be reliable. In our recent experiments, we have been able to crash 25-33% of the utility programs on any version of UNIX that was tested. This report describes these tests and an analysis of the program bugs that caused the crashes.

- Miller, Fredriksen, and So.

The Microsoft SDL fuzzing requirement states that an application with file handling code needs to consume 100,000 fuzzed files. This level of fuzz testing gives additional confidence that your application can handle maliciously corrupted files without failing due to buffer overflows or other potential security vulnerabilities.
void top(char input[4])
{
    int cnt = 0;
    if (input[0] == 'b') cnt++;
    if (cnt >= 1) crash();
}

One in $2^8$ chance!
The Fix: Whitebox Fuzz Testing

Fuzz Testing

plus

Dynamic Test Generation
Dynamic Test Generation

- Trace dynamic execution of program
- Capture symbolic path condition
- Create symbolic formula for new path
- Solve new path condition
- Generate new test case from solution
- DART (Godefroid-Klarlund-Sen 2005)
  EGT (Cadar-Engler 2005)
  EFFIGY (static test generation, King 1976)
void top(char input[4])
{
    int cnt = 0;
    if (input[0] == 'b') cnt++;
    if (cnt >= 1) crash();
}
Dynamic Test Generation

input = "good"

void top(char input[4])
{
    int cnt = 0;
    if (input[0] == 'b') cnt++;
    if (cnt >= 1) crash();
}

Path Constraint:
input[0] != 'b'
Dynamic Test Generation

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

New constraint:

```c
input[0] == 'b'
```
Dynamic Test Generation

New input = “bood”

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

New constraint:
input[0] == 'b'
```
**Dynamic Test Generation**

**New input = “bood”**

```c
void top(char input[4])
{
    int cnt = 0;
    if (input[0] == ‘b’) cnt++;
    if (cnt >= 1) crash();
}
```

**New constraint:**

```c
input[0] == ‘b’
```
Challenge: Scale!

- First generation tools exciting, small programs
  - EGT (2KLOC), CUTE (2KLOC), DART (30KLOC)
- Depth-first search limits scaling
  - Symbolic execution expensive
- Main Problem: Scale to larger programs!
What Search Strategy Should We Use for Larger Programs?

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();
}

I_0 != 'b'
I_1 != 'a'
I_2 != 'd'
I_3 == '!'
Depth-First Search Requires Many Symbolic Executions of the Program

```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

Time to create symbolic trace: 25m30s
Symbolic branches/trace: ~1000
Time/branch to solve, generate new test, check for crashes: ~1s

Therefore, solve+check all branches for each trace!
New Strategy: 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();
}

“Generation 1” test cases = {bood, gaod, godd, goo!}
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();
}
Active Property Checking

- Direct search towards property violations
  - Buffer overflow or underflow
  - Passing NULL parameters
  - Division by zero
- Check many properties simultaneously!
- Divide solver queries into two types
  - “Coverage-seeking”
  - “Bug-seeking”
Key Properties: Integer Bugs

- Number 2 cause of vendor security advisories
- Underflow/Overflow
- Value conversions
- Signed/Unsigned conversion bugs
- Poor fit with traditional runtime, static analysis
  - Static analysis: false positives
  - Runtime analysis: “benign” overflow problem
void bad(int x, char * src, char * dst)
{
    if (x > 800)
    {
        return;
    }
    else
    {
        copy_bytes(x, src, dst);
    }
}
void bad(int x, char * src, char * dst)
{
    if (x > 800)
    {
        return;
    }
    else
    {
        copy_bytes(x, src, dst);
    }
}

What if x == -1?
void bad(int x, char * src, char * dst) {
    if (x > 800) \(-1 > 800\) ? No! {
        return;
    }
    else {
        copy_bytes(x, src, dst);
    }
}
void bad(int x, char * src, char * dst)
{
    if (x > 800)
    {
        return;
    }
    else
    {
        copy_bytes(unsigned int x,...
        copy_bytes(x, src, dst);
    }
}
Signed/Unsigned Conversion Bugs

void bad(int x, char * src, char * dst) {
    if (x > 800) {
        return;
    }

    else {
        copy_bytes(\texttt{unsigned int} x, ...
        copy_bytes(x, src, dst);
    
    }  
    
    Copy a few more than 800 bytes..!
void bad(int x, char * src, char * dst) {
    if (x > 800) {
        return;
    }
    else {
        copy_bytes(x, src, dst);
    }
}
Idea:
1. **Keep track of type** for every tainted program value
2. Use solver to **force values with type “Bot” to equal -1**
   Developed methods to **infer types over long binary traces.**
Dynamic Test Generation Architecture

Check for Crashes → Trace Program → Gather Constraints → Solve Constraints

Input0
Trace File
Constraints
Input1
Input2
...
InputN
Putting It All Together

SAGE

SmartFuzz

Microsoft

The University of California, Berkeley
SAGE Initial Experiences

• Released internally in Microsoft April 2007

• Dozens of new security bugs
  • Most missed by blackbox fuzz, static analysis
  • “security critical, severity 1, priority 1”

• Several teams use SAGE daily

• Credit is due to the entire SAGE team and users:

  **CSE:** Michael Levin (DevLead), Christopher Marsh, Dennis Jeffries (intern’06), Adam Kiezun (intern’07); Mgmt: Hunter Hudson, Manuvir Das,… (+ symbolic exec. engine Nirvana/iDNA/TruScan contributors)

  **MSR:** Patrice Godefroid, David Molnar (intern’07)
  (+ constraint solvers Disolver and Z3 contributors)
Plus work of many beta users who found and filed most of these bugs!
ANI Parsing - MS07-017

RIFF...ACONLIST
B...INFOINAM....
3D Blue Alternative v1.1..IART....

................
1996..anih$...$

................
rate...........

........... seq ..

_LIST....framic on..........

RIFF...ACONB
B...INFOINAM....
3D Blue Alternative v1.1..IART....

................
1996..anih$...$

................
rate...........

........... seq ..

__________________________

One in $2^{32}$ chance!
Zero to Crash in 10 Generations

• Starting with 100 zero bytes ...

• SAGE generates a crashing test for media 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 media 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 media 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 media parser:

```plaintext
<table>
<thead>
<tr>
<th>Generation 3</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
</tr>
<tr>
<td>00000000h: 52 49 46 46 3D 00 00 00 ** ** ** 20 00 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 media 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 media 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 media 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 73 74 72 68 00 00 00 00 76 69 64 73 ; ...strh...vids
00000040h:  00 00 00 00 00 73 74 72 66 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 media parser:

Starting with 100 zero bytes …

Generation 7
Zero to Crash in 10 Generations

• Starting with 100 zero bytes ...

• SAGE generates a crashing test for media parser:

Generation 8
Zero to Crash in 10 Generations

- Starting with 100 zero bytes ...

- SAGE generates a crashing test for media parser:

<table>
<thead>
<tr>
<th>Generation</th>
<th>Hexadecimal Values</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>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>
</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>
</tr>
<tr>
<td>00000030h:</td>
<td>00 00 00 00 73 74 72 68 00 00 00 00 76 69 64 73</td>
<td>....strh....vids</td>
</tr>
<tr>
<td>00000040h:</td>
<td>00 00 00 00 73 74 72 66 00 00 00 00 28 00 00 00</td>
<td>....strf....(...</td>
</tr>
<tr>
<td>00000050h:</td>
<td>00 00 00 00 00 00 00 00 00 00 00 00 01 00 00 00</td>
<td>................</td>
</tr>
<tr>
<td>00000060h:</td>
<td>00 00 00 00</td>
<td>................</td>
</tr>
</tbody>
</table>

Generation 9
Zero to Crash in 10 Generations

• Starting with 100 zero bytes ...

• SAGE generates a crashing test for media 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 72 72 66 B2 75 76 3A 28 00 00 00 ; ....strf²uv:...
00000050h: 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 – crashes the program!

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

- Implementation for x86 Linux binary programs
- Valgrind framework, STP solver
- UC Berkeley
- Available on Sourceforge
  - http://www.sf.net/projects/catchconv/
## SAGE and SmartFuzz Scale

<table>
<thead>
<tr>
<th>App Tested</th>
<th>SLOC</th>
<th># x86Inst / Trace</th>
</tr>
</thead>
<tbody>
<tr>
<td>ANI</td>
<td>NA</td>
<td>2,066,087</td>
</tr>
<tr>
<td>Media 1</td>
<td>NA</td>
<td>3,409,376</td>
</tr>
<tr>
<td>Media 2</td>
<td>NA</td>
<td>271,432,489</td>
</tr>
<tr>
<td>Media 3</td>
<td>NA</td>
<td>54,644,652</td>
</tr>
<tr>
<td>Media 4</td>
<td>NA</td>
<td>133,685,240</td>
</tr>
<tr>
<td>Compression</td>
<td>NA</td>
<td>480,435</td>
</tr>
<tr>
<td>Office 2007</td>
<td>NA</td>
<td>923,731,248</td>
</tr>
<tr>
<td>mplayer</td>
<td>723,468</td>
<td>159,500,373</td>
</tr>
<tr>
<td>ffmpeg</td>
<td>304,990</td>
<td>19,539,096</td>
</tr>
<tr>
<td>Exiv2</td>
<td>57,080</td>
<td>6,185,985</td>
</tr>
<tr>
<td>gzip</td>
<td>140,036</td>
<td>161,118</td>
</tr>
<tr>
<td>bzip2</td>
<td>26,095</td>
<td>746,219,573</td>
</tr>
<tr>
<td>convert (PNG&gt;GIF)</td>
<td>300,896</td>
<td>478,474,232</td>
</tr>
</tbody>
</table>
SmartFuzz Experiments

- Six Linux programs
  - mplayer, ffmpeg, convert, gzip, bzip2, exiv2
- Three seed files each
- For each program, each seed file
  - 24 hours with SmartFuzz
  - 24 hours with a blackbox fuzzer, zzuf

- Amazon Elastic Compute Cloud
  - 1.7 GB RAM machine for $0.10/hr
  - 7 GB RAM machine for $0.40/hr
Millions of test cases!
How do we find the “right” tests?
Valgrind Memcheck

Checks execution for:
- InvalidWrite
- InvalidRead
- Leak_DefinitelyLost
- UninitializedValue
- Overlap
- More errors...

Used regularly by dozens of projects, including Firefox and OpenOffice.
SUPERB TRUST 2008

We found 1,124,452 test cases with Valgrind errors and all we got was this lousy T-shirt!
SUPERB TRUST 2008

Does that mean 1,124,452 bugs?
Problem: Bug Bucketing

Bug # 1

Bug # 2

Bug # 3
First Try : Stack Hash

Program received signal SIGSEGV, Segmentation fault.
[Switching to Thread -1209591584 (LWP 16035)]
0x081e03db in dct36 (inbuf=0xbfffc290, o1=0x8705500, o2=0x8704300, wintab=0x40bb3a40,
   tsubuf=0xbff090) at mp3lib/dct36.c:169
169   MACRO(0);
(gdb) bt
#0 0x081e03db in dct36 (inbuf=0xbfffc290, o1=0x8705500, o2=0x8704300, wintab=0x40bb3a40,
   tsubuf=0xbff090) at mp3lib/dct36.c:169
#1 0x081e492e in do_layer3 (fr=0x8708640, single=-1) at mp3lib/layer3.c:1212
#2 0x081e6016 in MP3_DecodeFrame (hova=0x8979512
"&#65533;&#65533;&#65533;1&&65533;\236<g&65533;*E&65533;
;&&65533;<M3\005\211T&65533;n\177Z\v\&65533;\210\t021b\a\004k&
#65533;\a&#65533;o\026\b",
single=-1) at mp3lib/sr1.c:539
#3 0x080d83b5 in decode_audio (sh_audio=0x8977ea0, minlen=8192)
at libmpcdecs/dec_audio.c:383
#4 0x08075d3a in main (argc=3, argv=0xbff754) at mplayer.c:2044
First Try: Stack Hash

Program received signal SIGSEGV, Segmentation fault.

0x081e03db in dct36 (inbuf=0xbfffc290, o1=0x8705500, o2=0x8704300, wintab=0x40bb3a40, tsbuf=0xbfffb090) at mp3lib/dct36.c:169
169 MACRO(0);

(gdb) bt
#0 0x081e03db in dct36 (inbuf=0xbfffc290, o1=0x8705500, o2=0x8704300, wintab=0x40bb3a40, tsbuf=0xbfffb090) at mp3lib/dct36.c:169
#1 0x081e492e in do_layer (fr=0x8708640, single=-1) at mp3lib/layer3.c:1212
#2 0x081e6016 in MP3_DecodeFrame (hova=0x8979512 "&#65533;&#65533;&#65533;&#65533;
;&#65533;<M3\005\211T&#65533;
#65533;\a&#65533;o\026\b"

single=-1) at mp3lib/sr1.c:519
#3 0x080d83b5 in decode_audio (sh_audio=0x8977ea0, minlen=8192)
at libmpcodecs/dec_audio.c:383
#4 0x08075d3a in main (argc=3, argv=0xbfffb754) at mplayer.c:2044
Problem: Duplicate Bug Reports!

- SUPERB-TRUST students reported 110+ bugs
- Stack hash to avoid duplicates
- Developers marked 10+ as duplicates anyway
Approach: Fuzzy Stack Hash

Program received signal SIGSEGV, Segmentation fault.
[Switching to Thread -1209591584 (LWP 16035)]
0x081e03db in dct36 (inbuf=0xbfffc290, o1=0x8705500, wintab=0x40bb3a40,
tsbuf=0xbfffb090) at mp3lib/dct36.c:169
169 MACRO(0);
(gdb) bt
#0 0x081e03db in dct36 (inbuf=0xbfffc290, o1=0x8705500, wintab=0x40bb3a40,
tsbuf=0xbfffb090) at mp3lib/dct36.c:169
#1 0x081e492e in do_layer3 (fr=0x8708640, single=-1) at mp3lib/layer3.c:121
#2 0x081e6016 in MP3_DecodeFrame (hova=0x8979512
"&#65533;&#65533;&#65533;1&#65533;&#65533;\236<g&#65533;*E&#65533;
;&#65533;<M3\005\211T&#65533;\n\177Z\v\v&#65533;\:\\210\t021b&\a\004k&
#65533;\a&#65533;o\026\b",
single=-1) at mp3lib/sr1.c:539
#3 0x080d83b5 in decode_audio (sh_audio=0x8977ea0, minlen=8192)
    at libmpcodecs/dec_audio.c:383
#4 0x08075d3a in main (argc=3, argv=0xbfffb754) at mplayer.c:2044
Current metafuzz stats, last updated Thursday 12th of February 2009 03:29:01 AM:
2983 runs in database, total of 2528323 test files created, with 6824 distinct bug stack hashes over all runs.

See the [Premade VM Page](http://www.metafuzz.com/) for instructions on how to contribute your own results!

Current bugs with uploaded test cases (NOTE: this list is regenerated periodically and does not reflect an up to the minute list of test cases. See [http://www.metafuzz.com/testcases](http://www.metafuzz.com/testcases) for the most current list):

<table>
<thead>
<tr>
<th>Run UUID</th>
<th>Seq No</th>
<th>Stack Hash</th>
<th>Kind</th>
<th>Program</th>
<th>Status</th>
<th>Fuzz Type</th>
<th>Test Case</th>
<th>Submitter</th>
</tr>
</thead>
<tbody>
<tr>
<td>842855</td>
<td>0</td>
<td>2621982726</td>
<td>Leak_DefinitelyLost</td>
<td>nxt-bad</td>
<td>Not Yet</td>
<td>catchconv</td>
<td>Download?</td>
<td><a href="mailto:premade@metafuzz.com">premade@metafuzz.com</a></td>
</tr>
<tr>
<td>842855</td>
<td>0</td>
<td>134515870</td>
<td>InvalidRead</td>
<td>nxt-bad</td>
<td>Not Yet</td>
<td>catchconv</td>
<td>Download?</td>
<td><a href="mailto:premade@metafuzz.com">premade@metafuzz.com</a></td>
</tr>
<tr>
<td>842855</td>
<td>0</td>
<td>816337621</td>
<td>InvalidRead</td>
<td>nxt-bad</td>
<td>Not Yet</td>
<td>catchconv</td>
<td>Download?</td>
<td><a href="mailto:premade@metafuzz.com">premade@metafuzz.com</a></td>
</tr>
<tr>
<td>842855</td>
<td>0</td>
<td>816599765</td>
<td>InvalidRead</td>
<td>nxt-bad</td>
<td>Not Yet</td>
<td>catchconv</td>
<td>Download?</td>
<td><a href="mailto:premade@metafuzz.com">premade@metafuzz.com</a></td>
</tr>
<tr>
<td>842855</td>
<td>0</td>
<td>816636885</td>
<td>InvalidRead</td>
<td>nxt-bad</td>
<td>Not Yet</td>
<td>catchconv</td>
<td>Download?</td>
<td><a href="mailto:premade@metafuzz.com">premade@metafuzz.com</a></td>
</tr>
<tr>
<td>980901</td>
<td>0</td>
<td>816637091</td>
<td>InvalidRead</td>
<td>nxt-bad</td>
<td>Not Yet</td>
<td>catchconv</td>
<td>Download?</td>
<td><a href="mailto:premade@metafuzz.com">premade@metafuzz.com</a></td>
</tr>
<tr>
<td>496868</td>
<td>0</td>
<td>1495414164</td>
<td>Leak_DefinitelyLost</td>
<td>samate</td>
<td>Not Yet</td>
<td>catchconv</td>
<td>Download?</td>
<td><a href="mailto:premade@metafuzz.com">premade@metafuzz.com</a></td>
</tr>
<tr>
<td>496868</td>
<td>0</td>
<td>4267069708</td>
<td>InvalidWrite</td>
<td>samate</td>
<td>Not Yet</td>
<td>catchconv</td>
<td>Download?</td>
<td><a href="mailto:premade@metafuzz.com">premade@metafuzz.com</a></td>
</tr>
</tbody>
</table>
SmartFuzz and zzuf Find Different Bugs (Under Fuzzy Stack Hash)
Both SmartFuzz and zzuf Find Bugs in Five out of Six Test Programs

<table>
<thead>
<tr>
<th></th>
<th>mplayer</th>
<th>ffmpeg</th>
<th>exiv2</th>
<th>gzip</th>
<th>convert</th>
</tr>
</thead>
<tbody>
<tr>
<td>SyscallParam</td>
<td>4</td>
<td>3</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>UninitCondition</td>
<td>13</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>UninitValue</td>
<td>0</td>
<td>3</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Overlap</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Leak DefinitelyLost</td>
<td>2</td>
<td>2</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Leak PossiblyLost</td>
<td>2</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>InvalidRead</td>
<td>1</td>
<td>2</td>
<td>4</td>
<td>6</td>
<td>2</td>
</tr>
<tr>
<td>InvalidWrite</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td><strong>Total</strong></td>
<td><strong>22</strong></td>
<td><strong>13</strong></td>
<td><strong>5</strong></td>
<td><strong>28</strong></td>
<td><strong>4</strong></td>
</tr>
<tr>
<td><strong>Cost per bug</strong></td>
<td>$1.30</td>
<td>$2.16</td>
<td>$5.76</td>
<td>$1.03</td>
<td>$3.60</td>
</tr>
</tbody>
</table>

No bugs found in bzip2.

Cost per bug at Amazon EC2 rates February 2009.
**SmartFuzz Beats zzuf on Two Out of Six Test Programs**

<table>
<thead>
<tr>
<th></th>
<th>mplayer</th>
<th>ffmpeg</th>
<th>exiv2</th>
<th>gzip</th>
<th>convert</th>
</tr>
</thead>
<tbody>
<tr>
<td>SyscallParam</td>
<td>4</td>
<td>3</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>UninitCondition</td>
<td>13</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>3</td>
</tr>
<tr>
<td>UninitValue</td>
<td>0</td>
<td>3</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Overlap</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Leak_DefinitelyLost</td>
<td>2</td>
<td>2</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Leak_PossiblyLost</td>
<td>2</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>InvalidRead</td>
<td>1</td>
<td>2</td>
<td>4</td>
<td>6</td>
<td>1</td>
</tr>
<tr>
<td>InvalidWrite</td>
<td>0</td>
<td>1</td>
<td>3</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td><strong>Total</strong></td>
<td>22</td>
<td>13</td>
<td>5</td>
<td>28</td>
<td>4</td>
</tr>
<tr>
<td><strong>Cost per bug</strong></td>
<td>$1.30</td>
<td>$2.16</td>
<td>$5.76</td>
<td>$1.03</td>
<td>$3.60</td>
</tr>
</tbody>
</table>

Note: 'NA' indicates not applicable.
## Bug-Seeking Queries Yield Bugs

<table>
<thead>
<tr>
<th></th>
<th>Queries</th>
<th>Test Cases</th>
<th>Bugs</th>
</tr>
</thead>
<tbody>
<tr>
<td>Coverage</td>
<td>588068</td>
<td>31121</td>
<td>19</td>
</tr>
<tr>
<td>ConversionNot32</td>
<td>4586</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Conversion32to8</td>
<td>1915</td>
<td>1377</td>
<td>3</td>
</tr>
<tr>
<td>Conversion32to16</td>
<td>16073</td>
<td>67</td>
<td>4</td>
</tr>
<tr>
<td>Conversion16Sto32</td>
<td>206</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>SignedOverflow</td>
<td>167110</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>SignedUnderflow</td>
<td>20198</td>
<td>21</td>
<td>3</td>
</tr>
<tr>
<td>UnsignedOverflow</td>
<td>164155</td>
<td>9280</td>
<td>3</td>
</tr>
<tr>
<td>MallocArg</td>
<td>30</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>SignedUnsigned</td>
<td>125509</td>
<td>6949</td>
<td>5</td>
</tr>
</tbody>
</table>
Theory Plus Practice

Theory of Symbolic Execution, Dynamic Test Generation → New software testing method → Scalable, Practical Tools
What Next?

Write Bug → Find Bug → Report Bug → Fix Bug → Write Bug
Thank you!
Questions?

dmolnar@eecs.berkeley.edu