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

Energy

Communication

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 ½ m 3 m

Intended Read Range

Deployment : Library RFID

Hello?

Unique ID (bar code)
Some vendors : title, author

Key Problems:

1. If title and author present, learn what someone reads.
2. "Hotlisting" : go to library, recognize book later.
3. Many deployed tags have unique IDs "baked in" to low-level collision avoidance protocols.

Deployment : e-Passports

Hello

Name, passport #,
Digital photograph,
nationality

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.
4. “U.S. citizen detector.”

Deployment : e-Passports

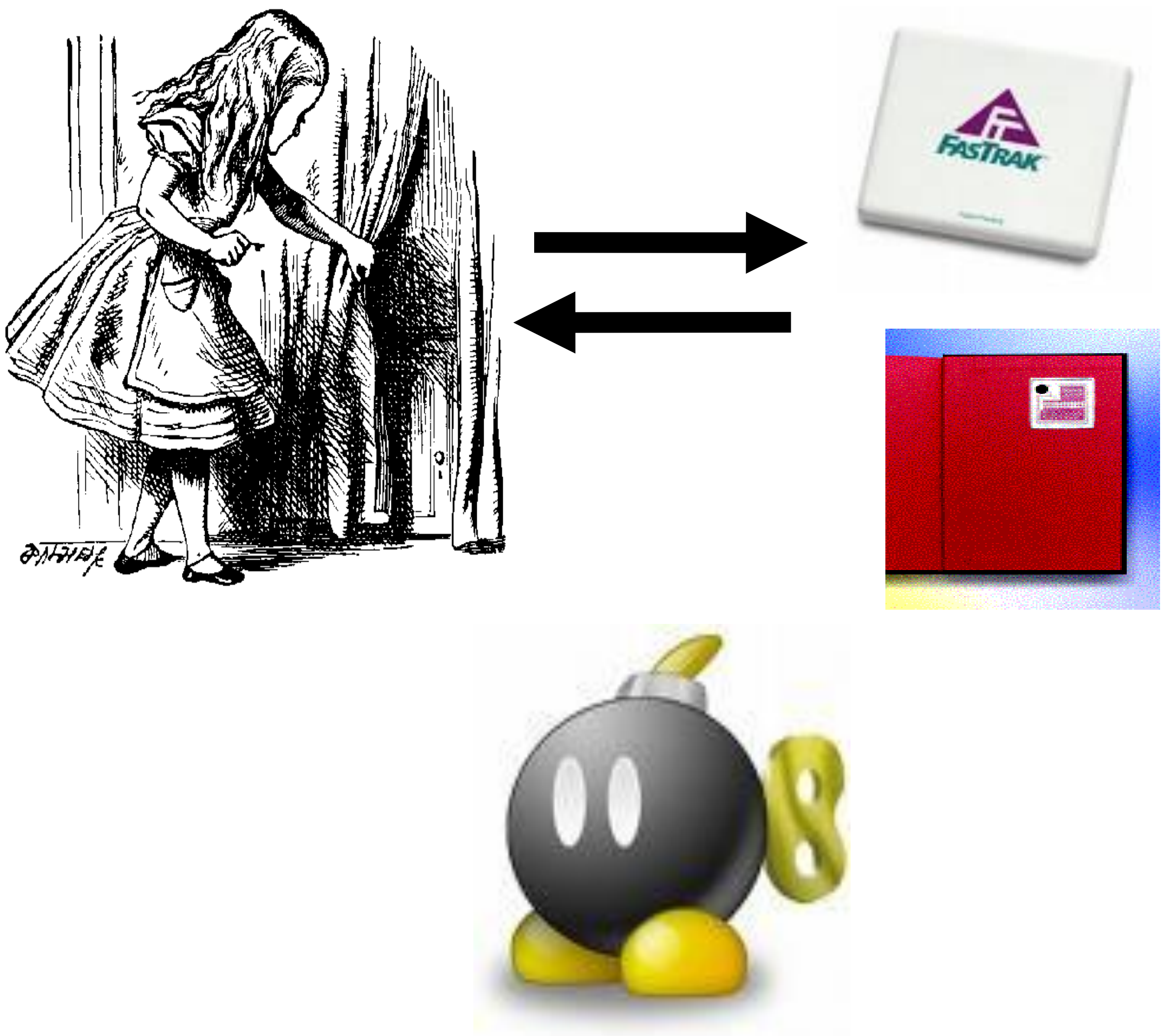
Hello

Encrypted (Name,...)

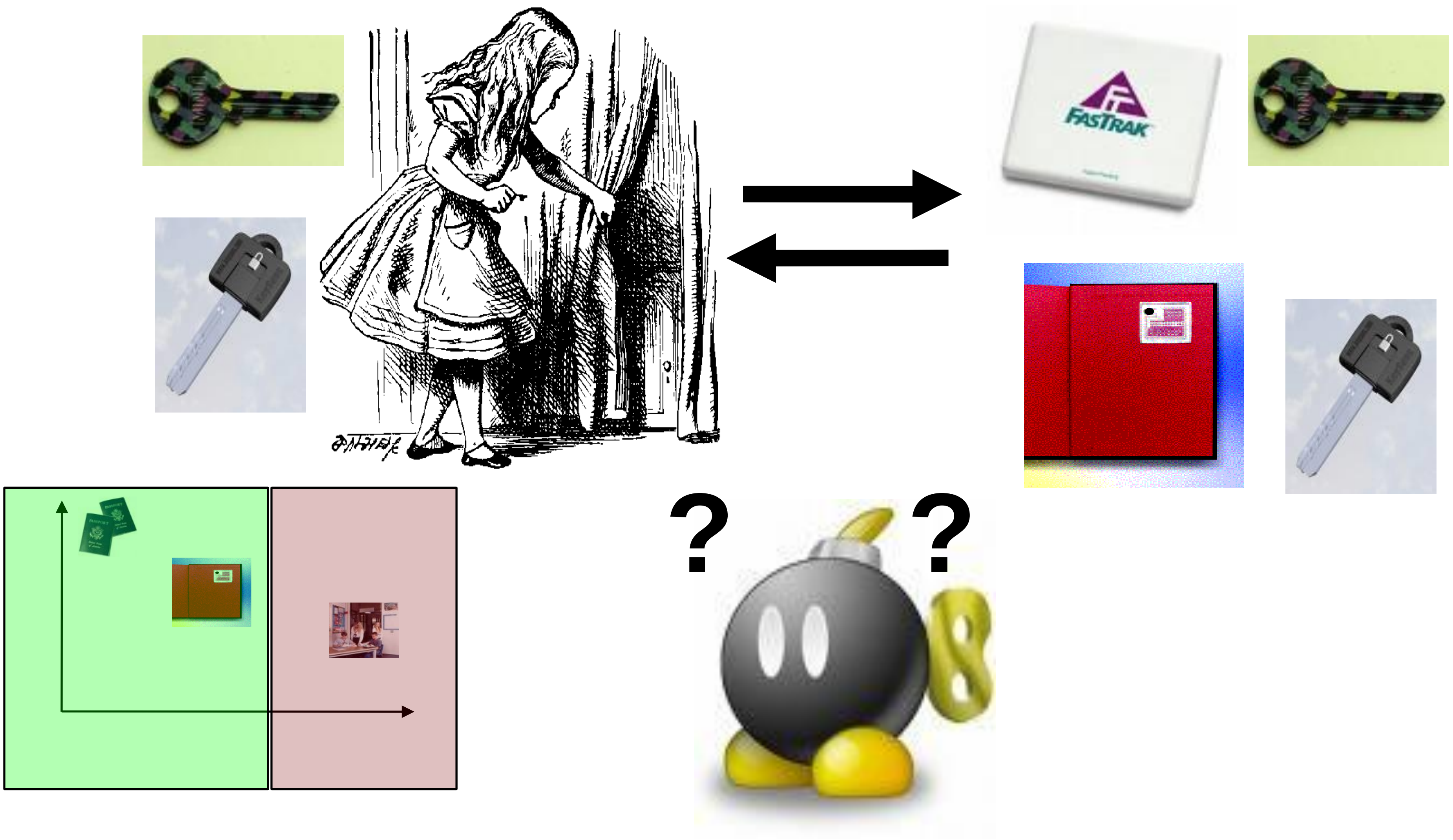
After our report, 2,400 negative public comments, and public outcry, U.S. State Department adds encryption to e-passports. Reader obtains encryption key by optically scanning the inside cover of the e-passport.

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


N

N’, F( ,N, N’)


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.

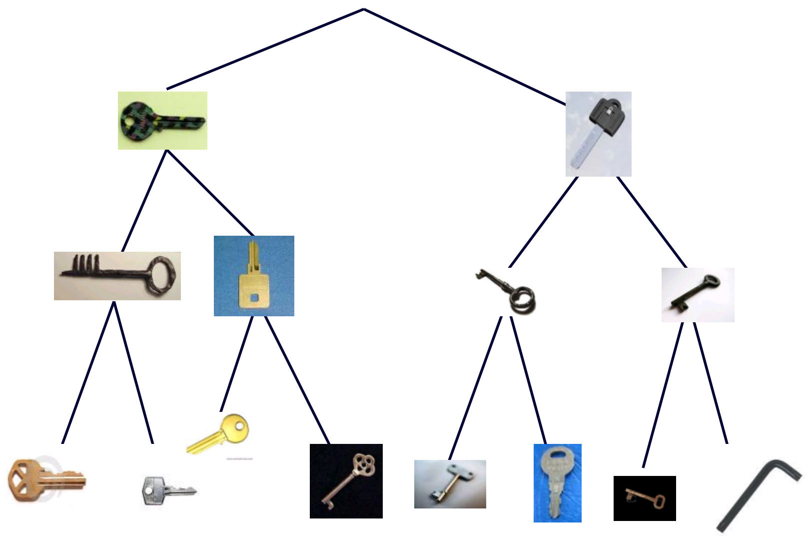
A Flawed Attempt At Scaling

N

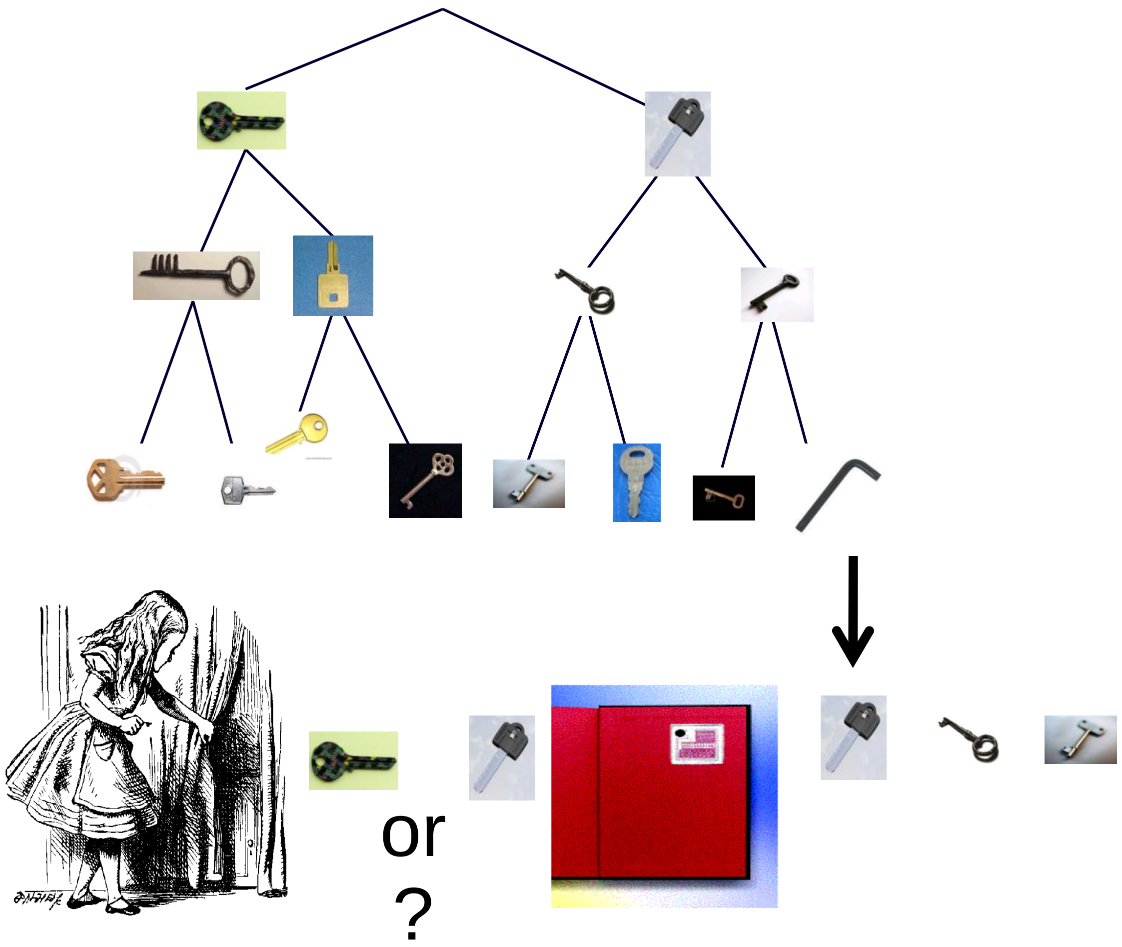
  ID,N', F( ,N, N')  

1. **Authentication.** Cannot predict F( ,N, N').
2. **Scaling.** Alice looks up key corresponding to ID, O(1).
3. **Privacy.** None! Tag sends ID in the clear.

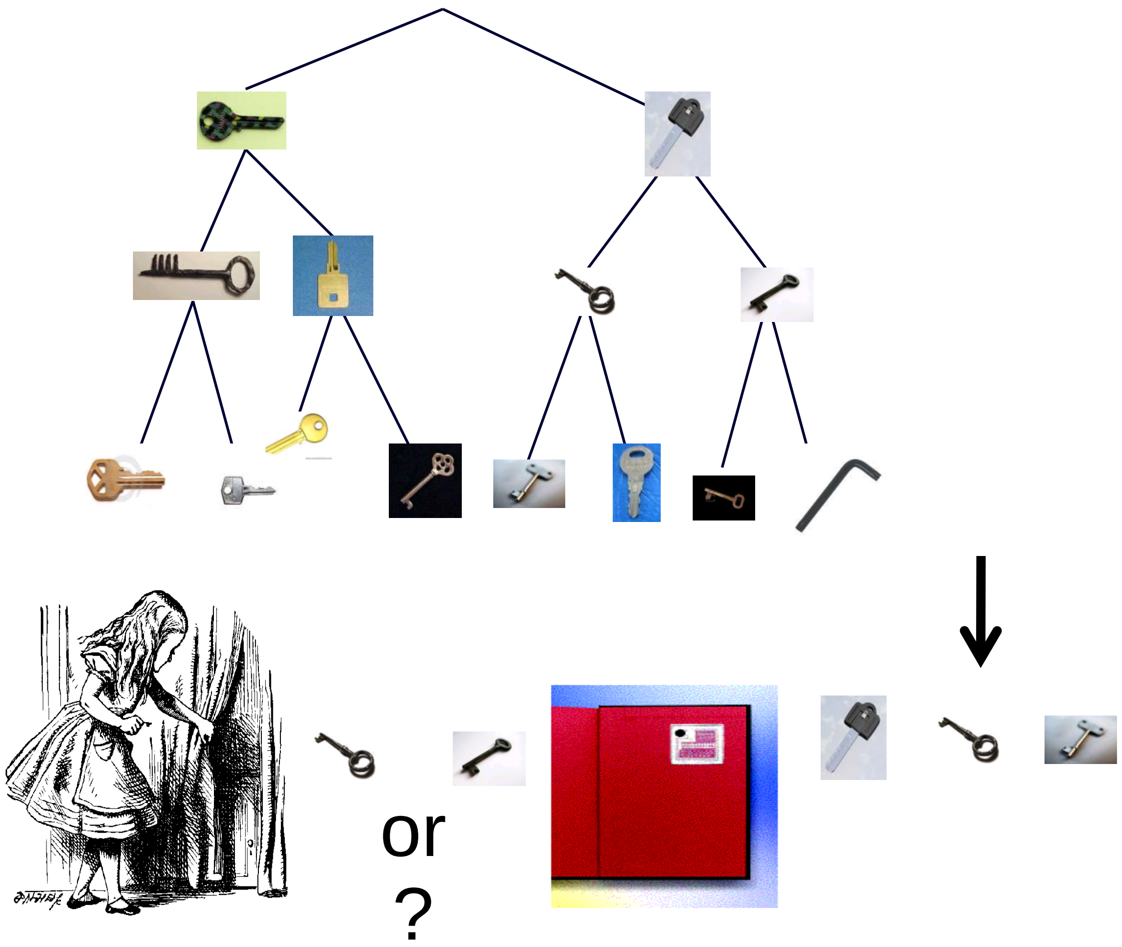
Scaling : A Key Tree



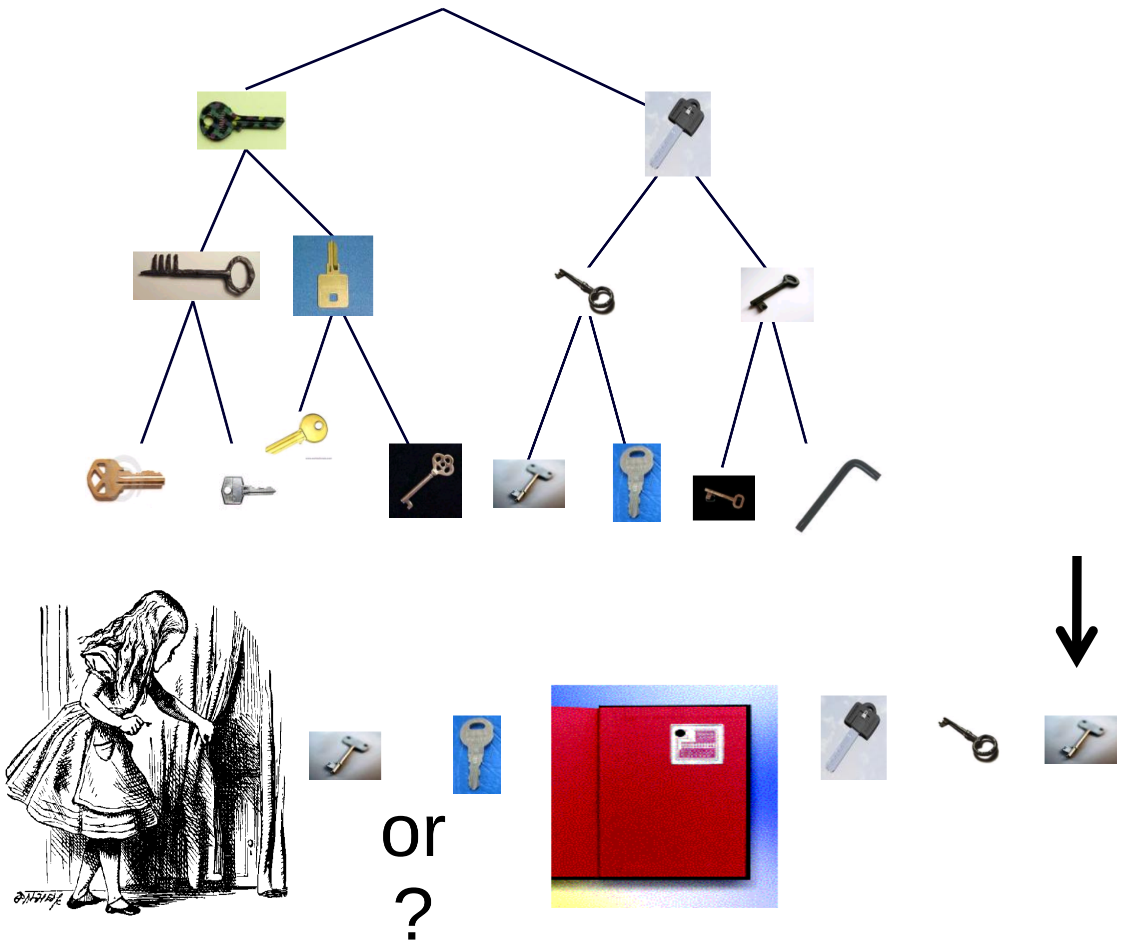
Scaling : A Key Tree



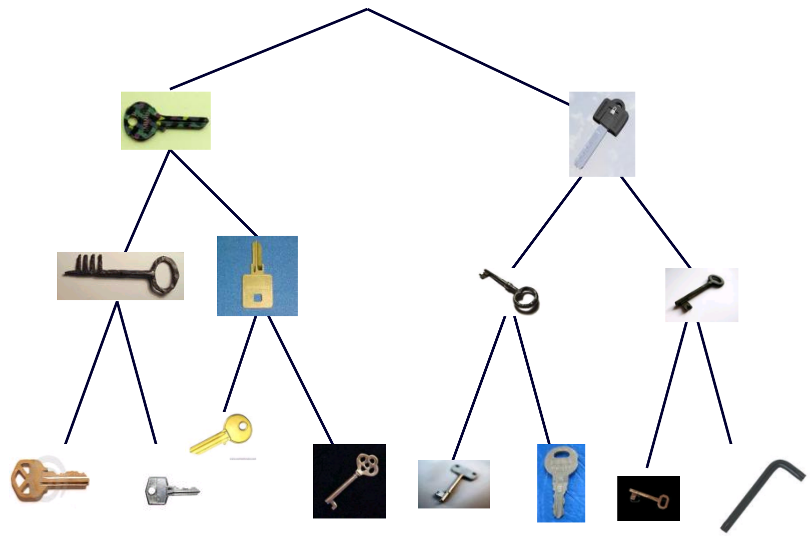
Scaling : A Key Tree



Scaling : A Key Tree



Scaling : A Key Tree



Book
4!

1. **Authentication.** Can’t predict F(key, N, 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:

Quantity	Tree Scheme	Naive	Precomputation
Reader Time	O(log T)	O(T)	O(T^2/3)
Reader Space	O(1)	O(1)	O(T^2/3)
Tag Time	O(log T)	O(1)	O(1)
Tag Storage	O(log T)	O(1)	O(1)
Communication	O(log T)	O(log T)	O(log T)

Concrete numbers for our tree scheme + optimizations in paper:

Quantity	
Number of Tags (T)	2^{20} = 1,048,576
Reader Time	2048 evaluations of F
Reader Space	128 bits
Tag Time	4 evaluations of F
Tag Space	192 bits
Communication	168 bits

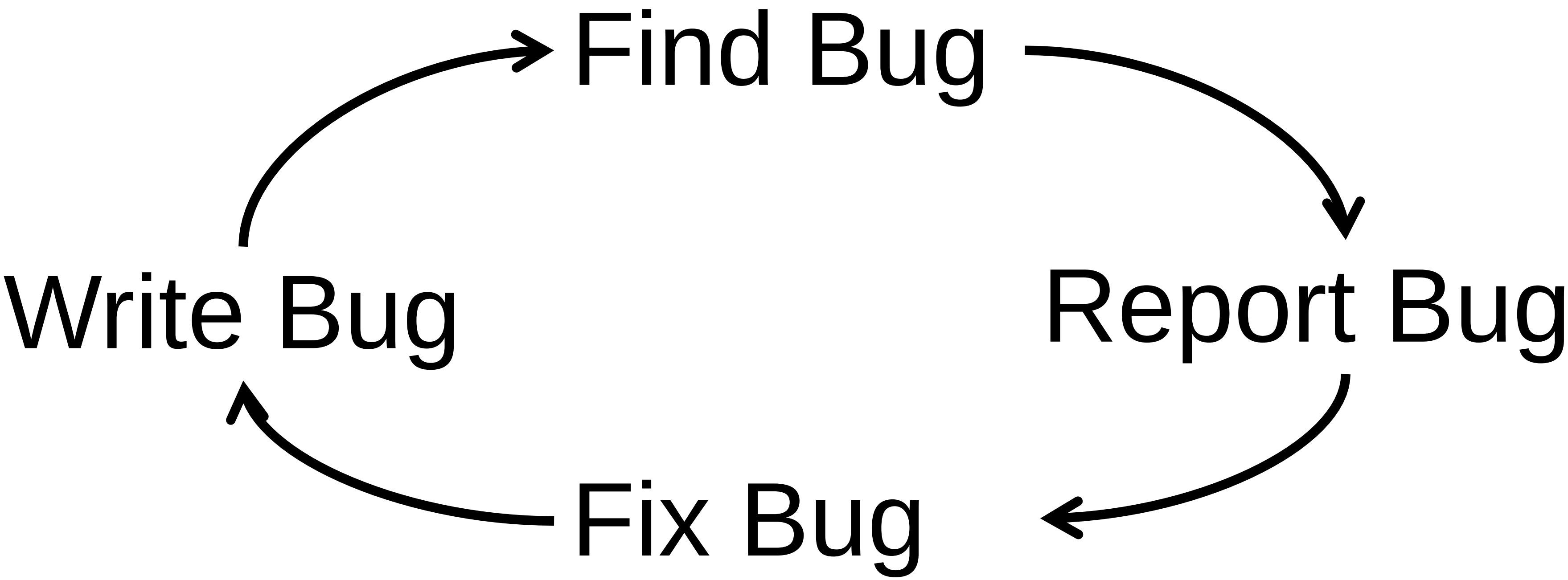
Theory Plus Practice

Security Analysis of Practical Deployments → Fundamental Problem of Scaling Private Authentication → Novel Algorithm for Solution

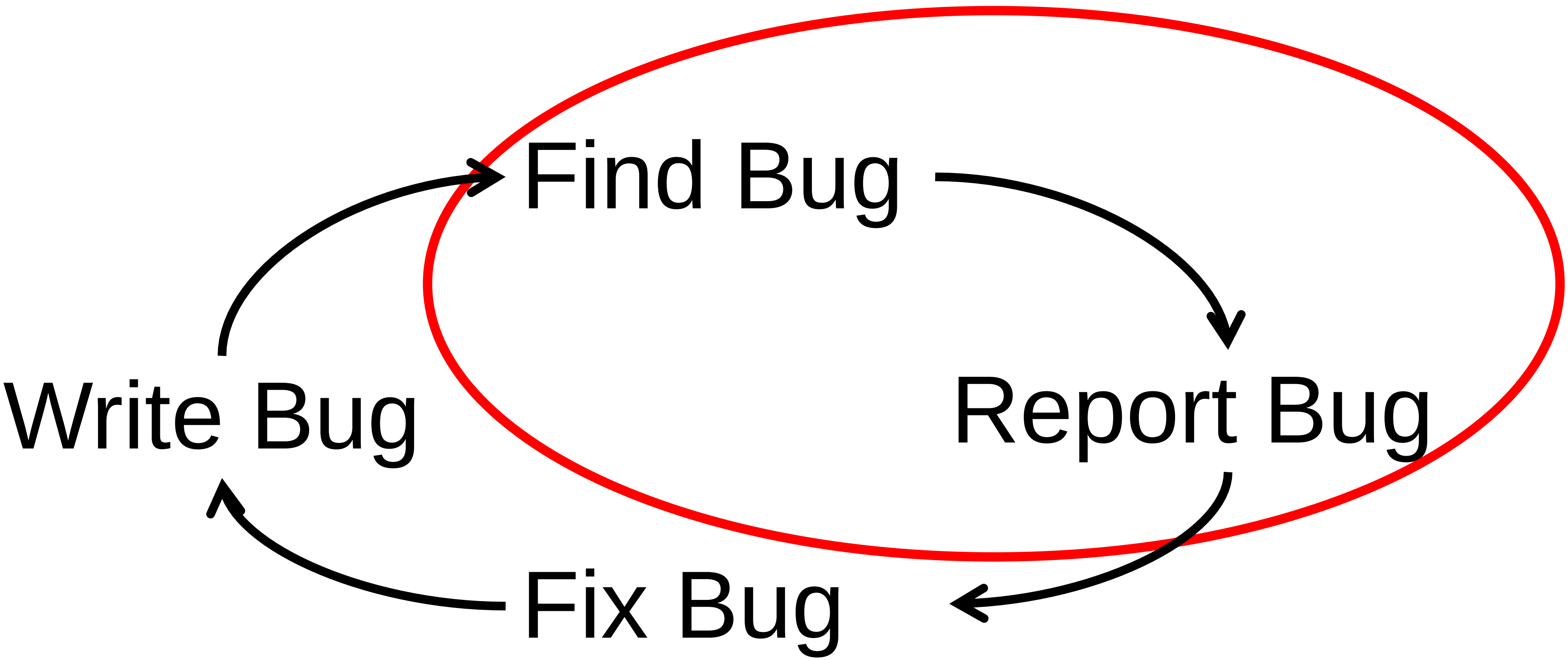
Software Security
-or-
Looking For High Value Bugs

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

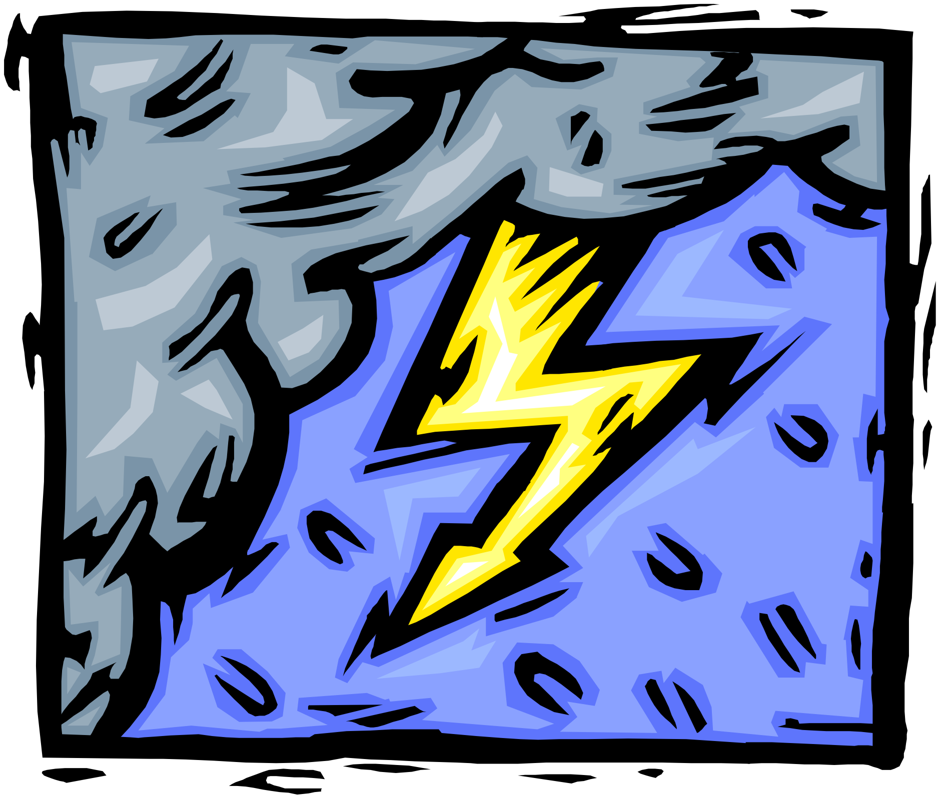


The “Bug Cycle”



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

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

NoA 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.

Key Limitation : “Unlikely” Paths

```
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)

Dynamic Test Generation

`input = “good”`

```
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

`input = "good"`

```
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”

```
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"

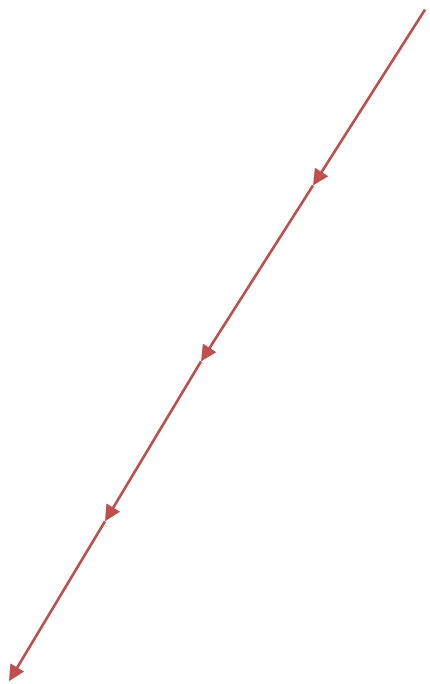
```
void top(char input[4])
{
    int cnt = 0;
    if (input[0] == 'b') cnt++;
    if (cnt >= 1) crash();
}
```

New constraint:
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?

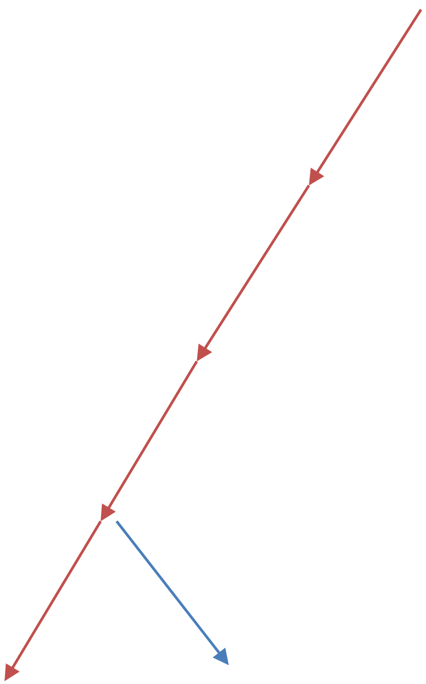


good

```
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();
}
```

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

Initial Choice: Depth-First Search



good 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();
}
```

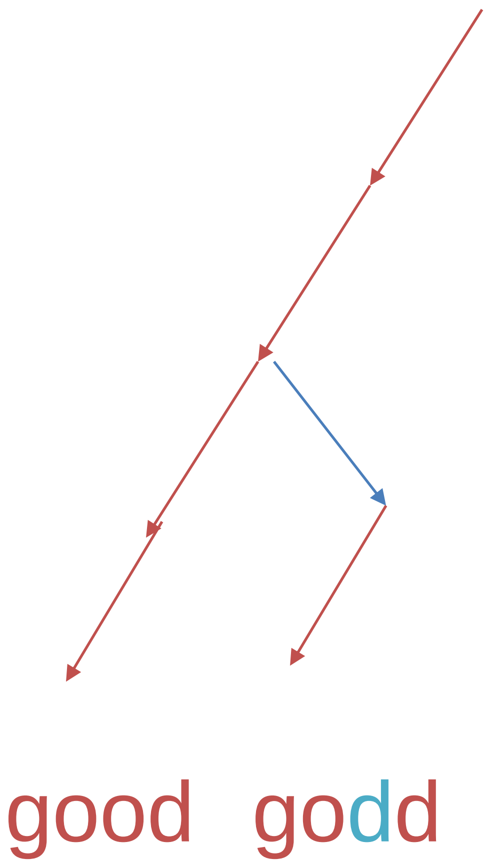
I_0 != ‘b’

I_1 != ‘a’

I_2 != ‘d’

I_3 == ‘!’

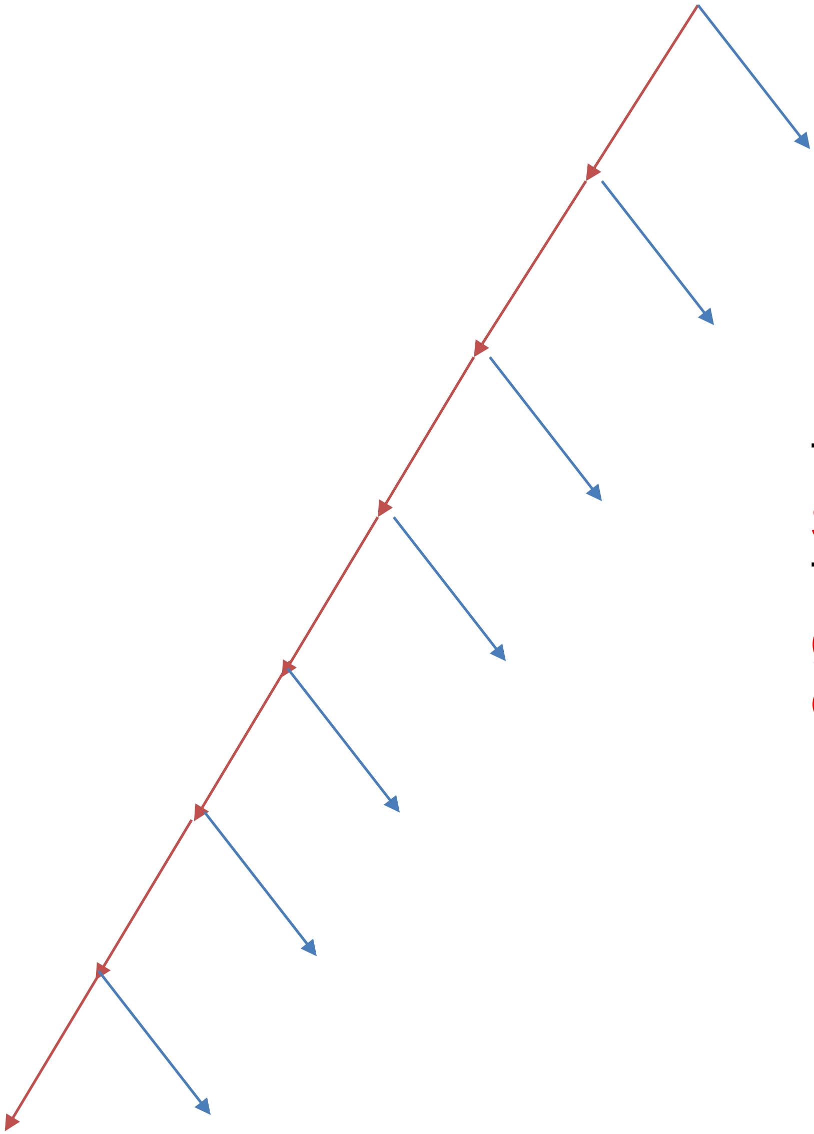
Depth-First Search Requires Many Symbolic Executions of the Program




```
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 != '!'

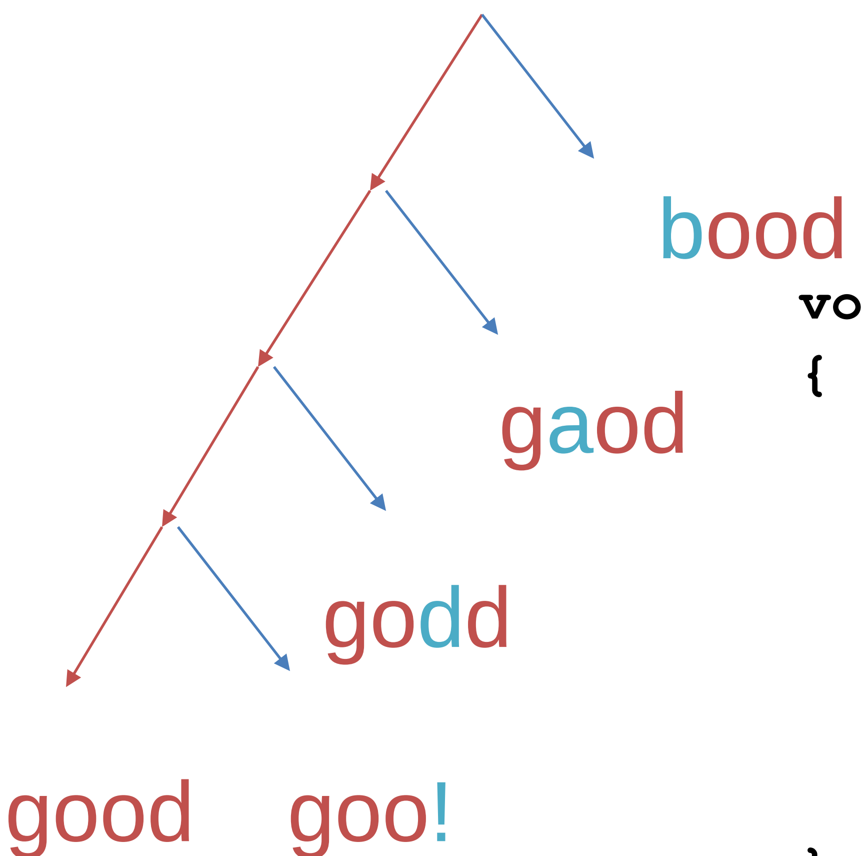
Key Idea : One Trace, Many Tests



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



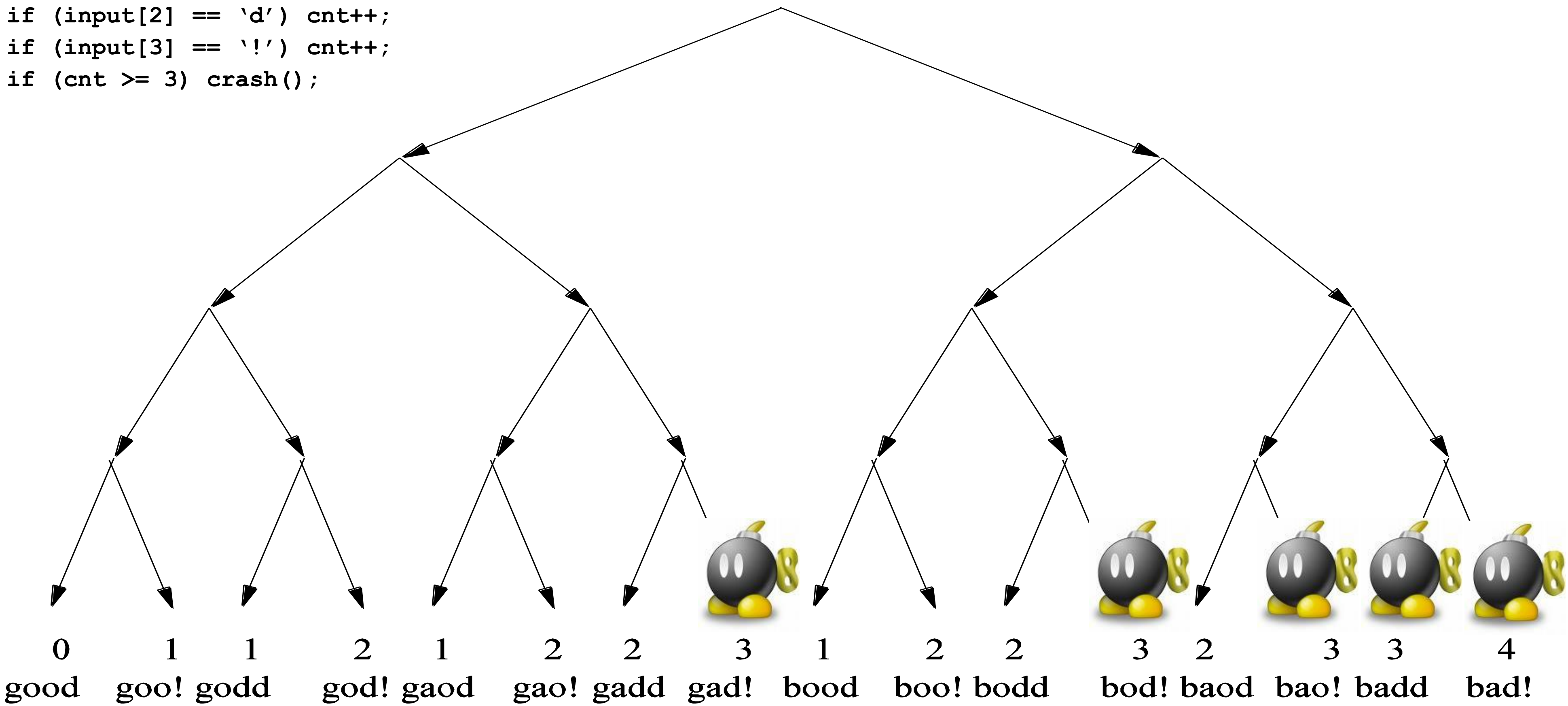

```
void top(char input[4])
{
    int cnt = 0;
    if (input[0] == ‘b’) cnt++;
    if (input[1] == ‘a’) cnt++;
    if (input[2] == ‘d’) cnt++;
    if (input[3] == ‘!’) cnt++;
    if (cnt >= 3) crash();
}
```

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

“Generation 1” test cases = {bood, gaod, godd, goo!}

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();
}
```



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

Signed/Unsigned Conversion Bugs

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

Signed/Unsigned Conversion Bugs

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

What if x == -1 ?

Signed/Unsigned Conversion Bugs

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

-1 > 800? **No!**

Signed/Unsigned Conversion Bugs

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

copy_bytes(**unsigned int** x,...

Signed/Unsigned Conversion Bugs

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

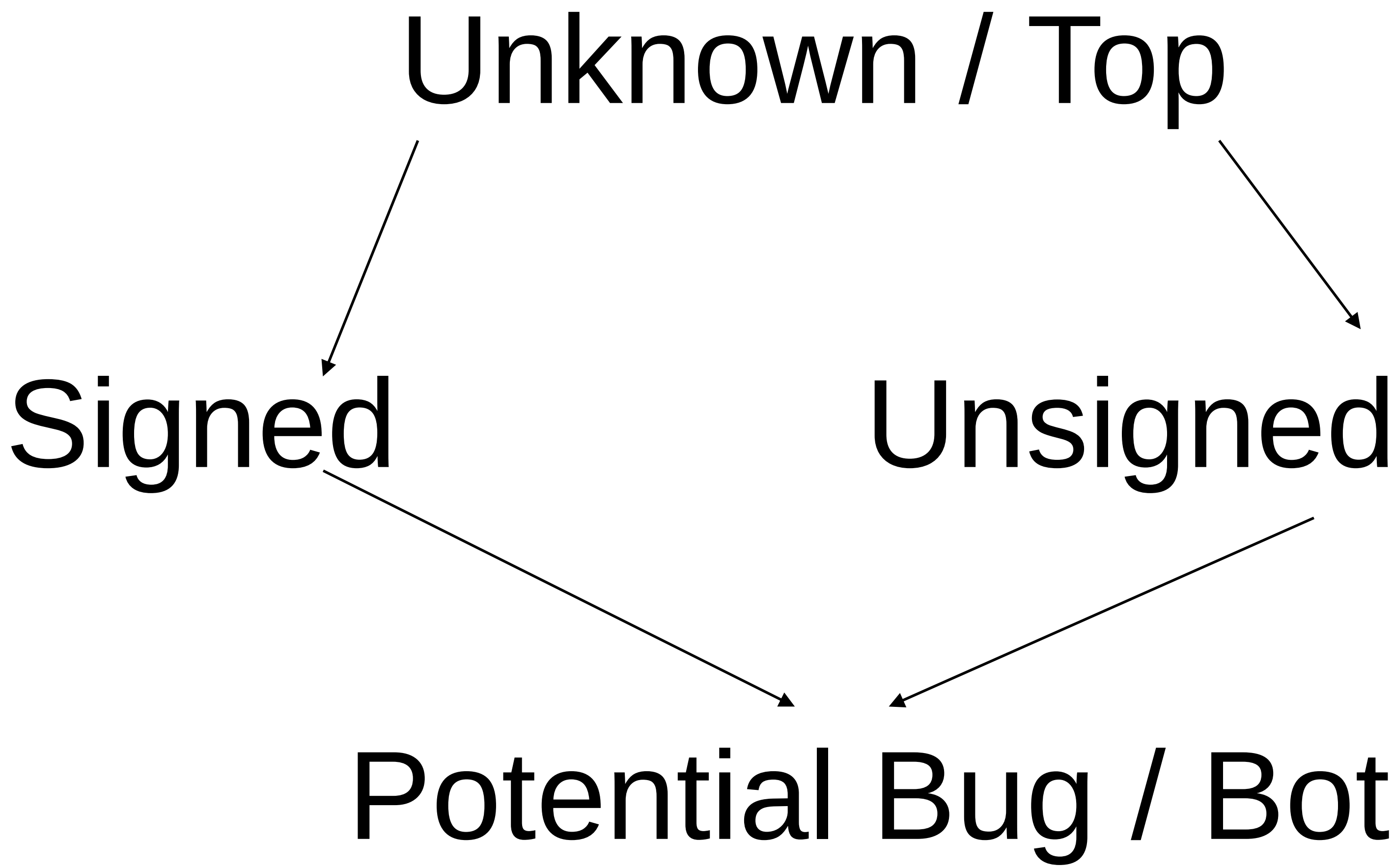
copy_bytes(**unsigned int** x,...

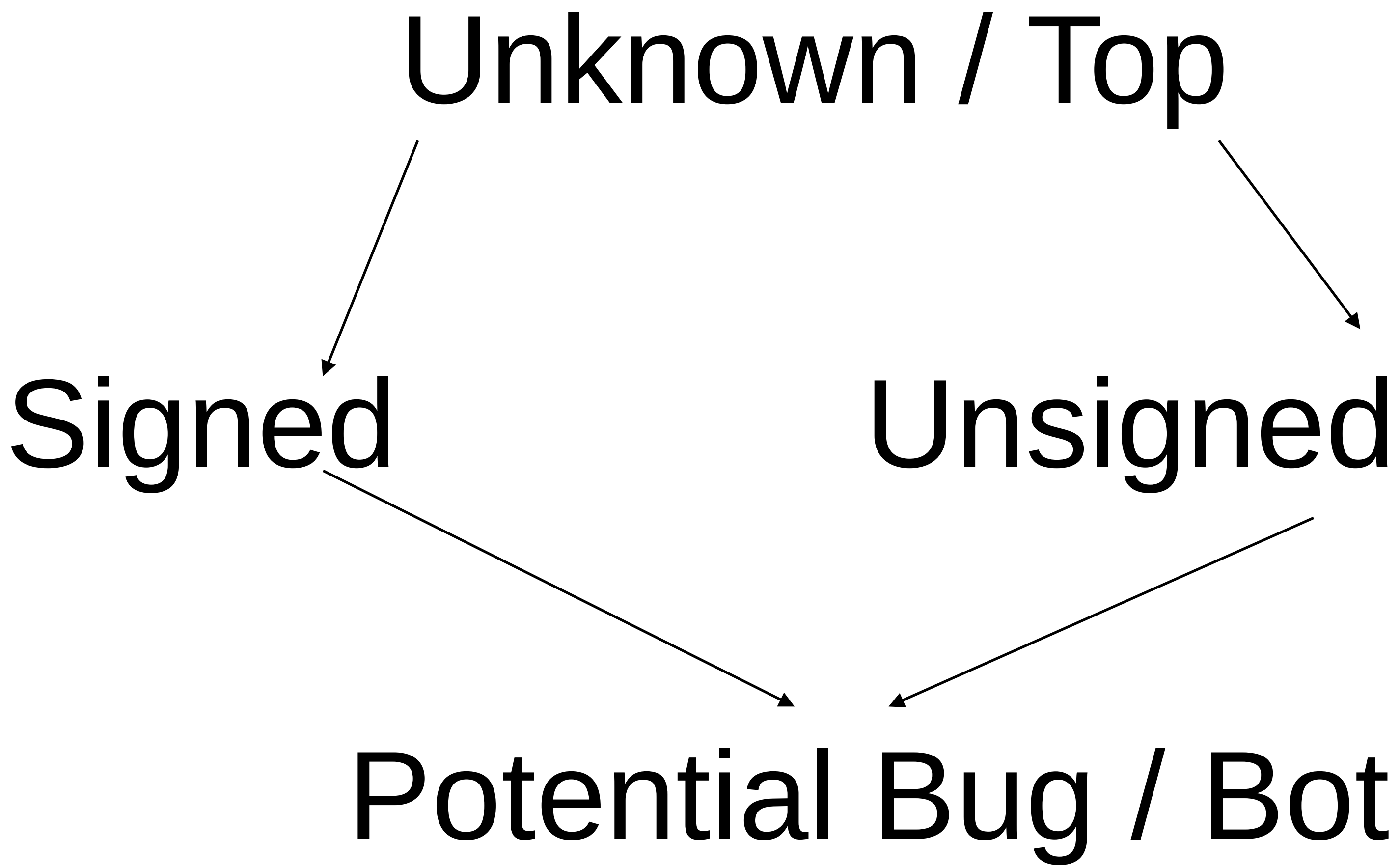
Copy a few more than 800 bytes..!

Signed/Unsigned Conversion Bugs

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

Bug pattern: treat value x as signed, then as unsigned or vice versa


Unknown / Top
Signed
Unsigned
Potential Bug / Bot

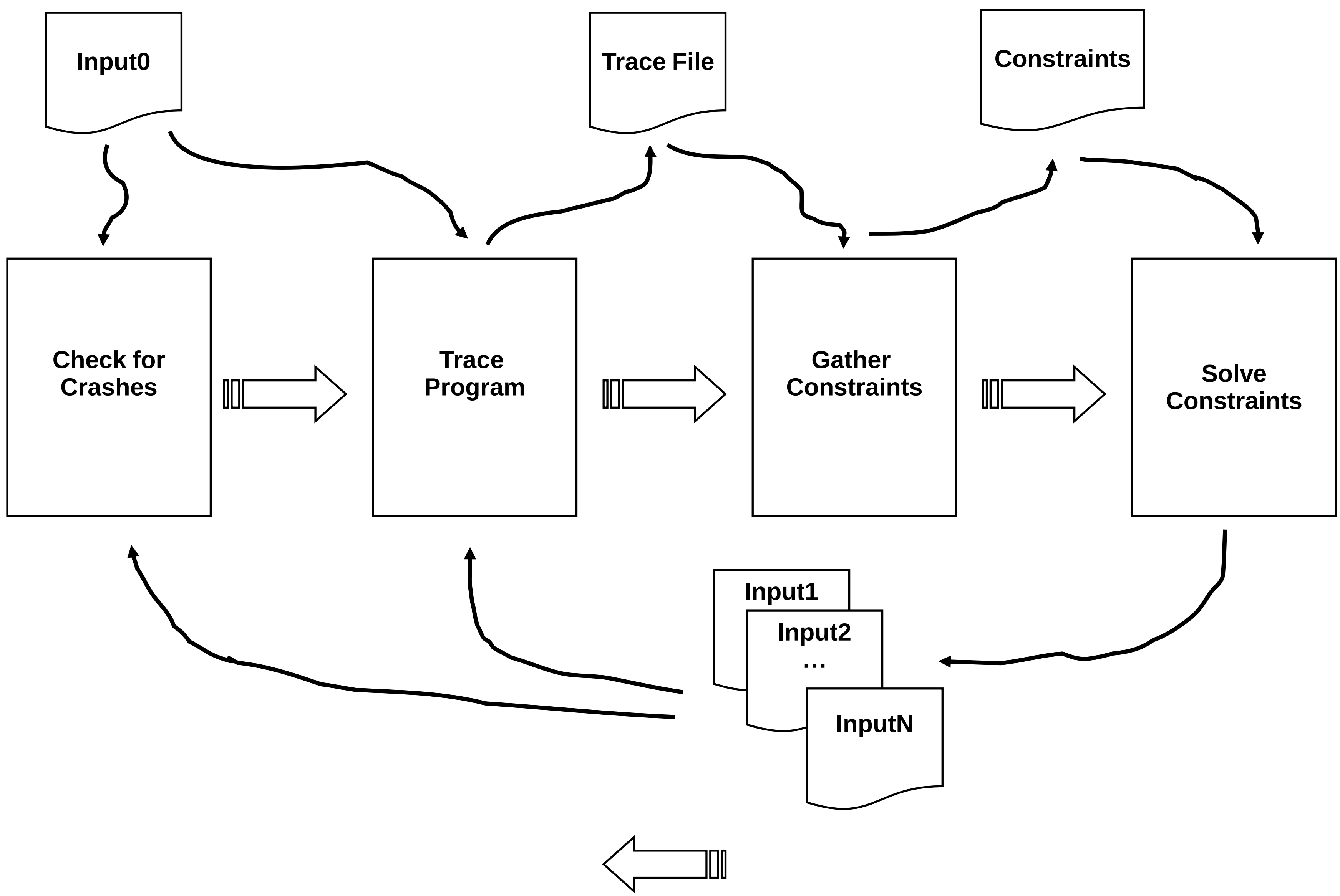


Idea:

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



Putting It All Together

SAGE

SmartFuzz

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 Alternat
e v1.1..IART....
................
1996..anih$...$.
................
................
..rate..........
..........seq ..
................
..LIST....framic
on........   ..
```

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

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:

```
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 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 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 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 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 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 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 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 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 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 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 – 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:// bit.ly/metafuzz-vm/
  - http://www.sf.net/projects/catchconv/

SAGE and SmartFuzz Scale

App Tested	SLOC	# x86Inst / Trace
ANI	NA	2,066,087
Media 1	NA	3,409,376
Media 2	NA	271,432,489
Media 3	NA	54,644,652
Media 4	NA	133,685,240
Compression	NA	480,435
Office 2007	NA	923,731,248

SAGE

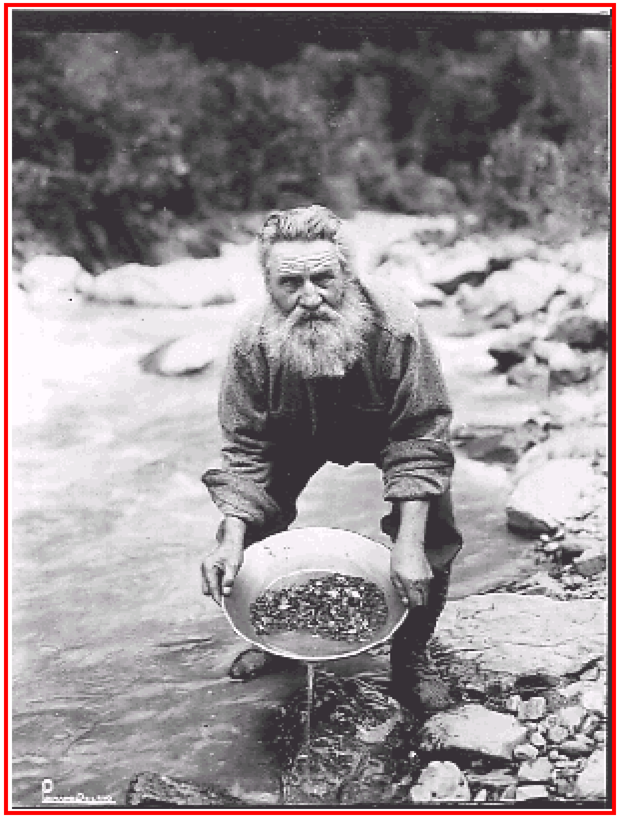
App Tested	SLOC	# x86Inst/ Trace
mplayer	723,468	159,500,373
ffmpeg	304,990	19,539,096
Exiv2	57,080	6,185,985
gzip	140,036	161,118
bzip2	26,095	746,219,573
convert (PNG>GIF)	300,896	478,474,232

SmartFuzz

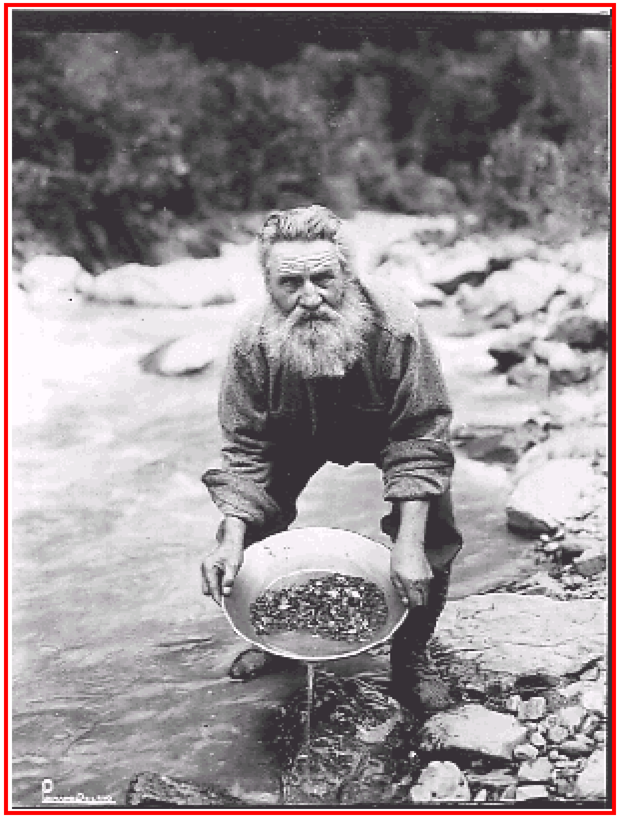
SmartFuzz Experiments

- 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
  - **7GB 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

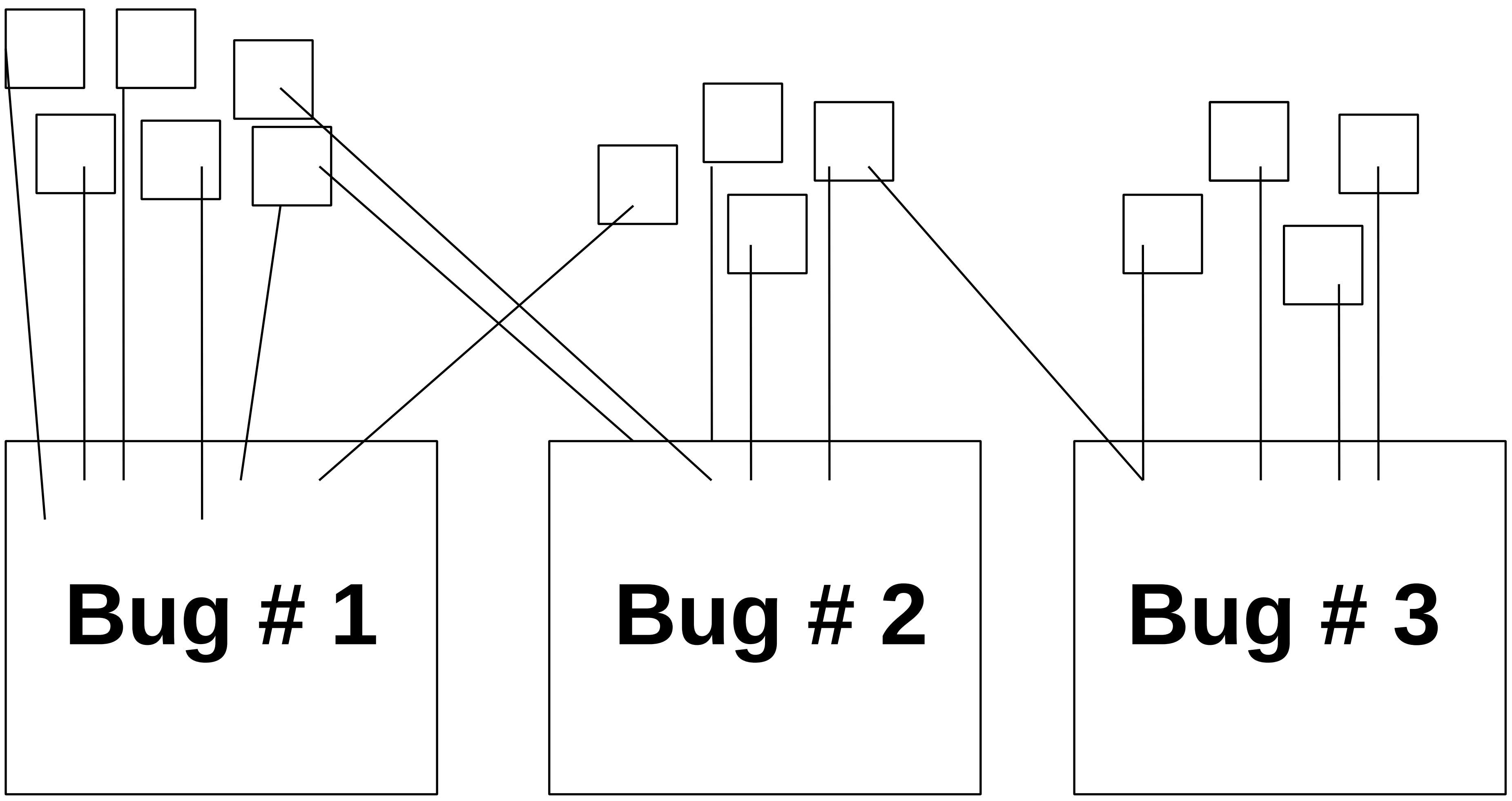
SUPERB TRUST 2008

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



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,
    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_layer3 (fr=0x8708640, single=-1) at mp3lib/layer3.c:1212
#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\t\021b&\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=0xbffff754) at mplayer.c:2044
```

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,
    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 mp3li
#1  0x081e492e in dc                                      3.c:1212
#2  0x081e6016 in M
    hova=0x8979512
"&#65533;&#65533;&#6553
;&#65533;<M3\005\211T&#6
#65533;\a&#65533;o\026\b".
single=-1) at mp3lib/sr1.c:5[illegible]9
#3  0x080d83b5 in decode_audio (sh_audio=0x8977ea0, minlen=8192)
    at libmpcodecs/dec_audio.c:383
#4  0x08075d3a in main (argc=3, argv=0xbffff754) at mplayer.c:2044
```

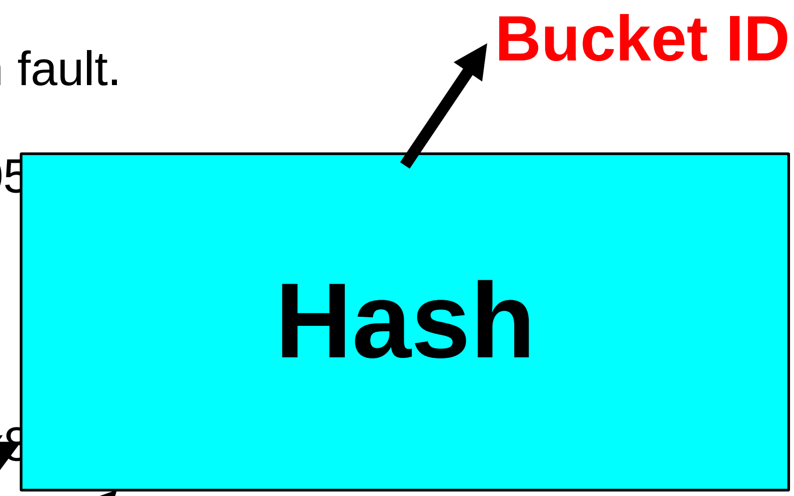
Hash

Bucket ID

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=0x8705
wintab=0x40bb3a40,
    tsbuf=0xbfffb090) at mp3lib/dct36.c:169
169   MACRO(0);
(gdb) bt
#0 0x081e03db in dct36 (inbuf=0xbfffc290, o1=0x8
wintab=0x40bb3a40,
    tsbuf=0xbfffb090) at mp3lib/**dct36.c:16**9
#1 0x081e492e in do_layer3 (fr=0x8708640, single=-1) at mp3lib**/layer3.c:121**2
#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\t\021b&\a\004k&
#65533;\a&#65533;o\026\b",
single=-1) at mp3lib/**sr1.c:53**9
#3 0x080d83b5 in decode_audio (sh_audio=0x8977ea0, minlen=8192)
    at libmpcodecs/dec_audio.c:383
#4 0x08075d3a in main (argc=3, argv=0xbffff754) at mplayer.c:2044

http://www.metafuzz.com

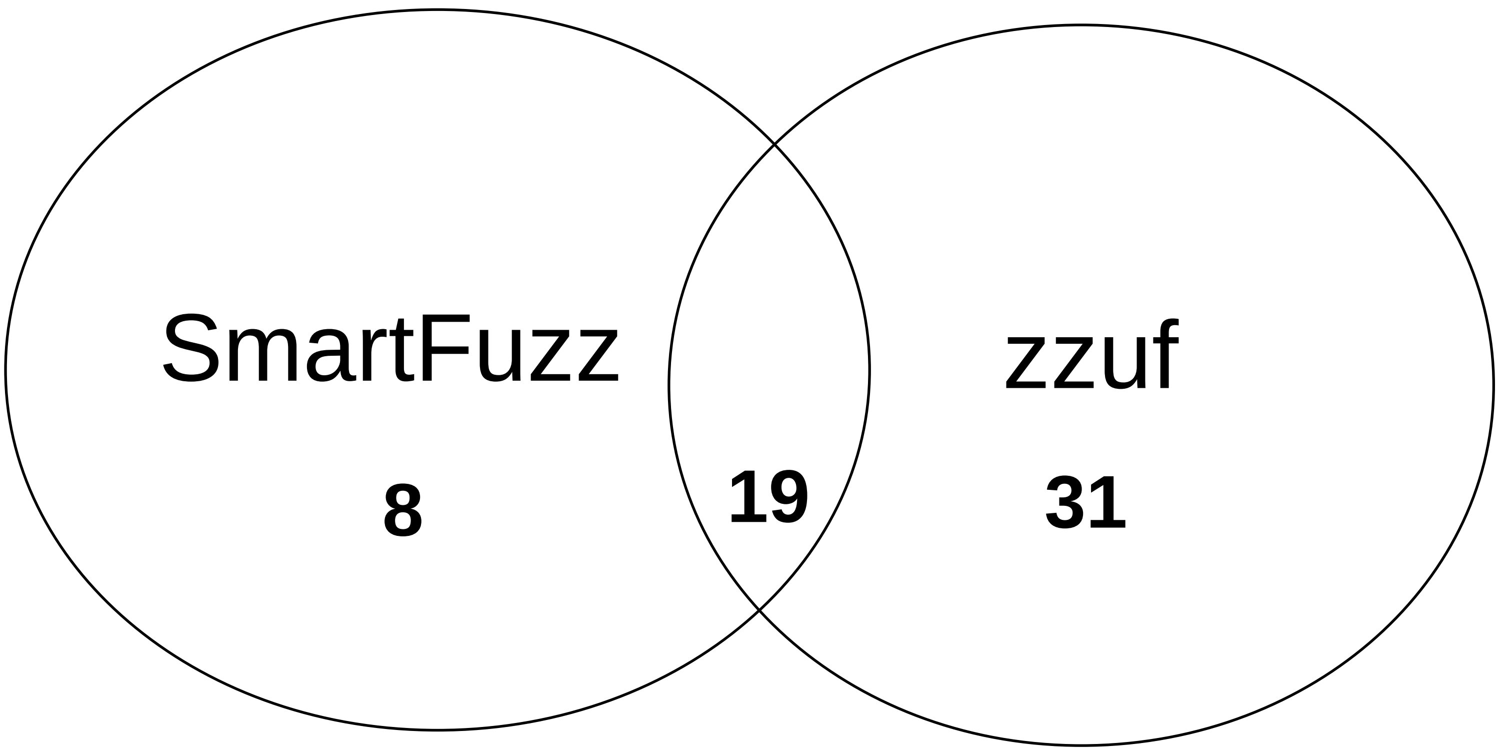
Current metafuzz stats, last updated Thursday 12th of February 2009 03:29:01 AM:
2983 runs in database, total of 2528323 test files created, with 6824 distinct bug stack hashes over all runs.

See the Premade VM Page 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 for the most current list):

Run UUID	Seq. No	Stack Hash	Kind	Program	Status	Fuzz Type	Test Case	Submitter
842855	0	2621982726	Leak_DefinitelyLost	nxt-bad	Not Yet	catchconv	Download?	premade@metafuzz.com
842855	0	134515870	InvalidRead	nxt-bad	Not Yet	catchconv	Download?	premade@metafuzz.com
842855	0	816337621	InvalidRead	nxt-bad	Not Yet	catchconv	Download?	premade@metafuzz.com
842855	0	816599765	InvalidRead	nxt-bad	Not Yet	catchconv	Download?	premade@metafuzz.com
842855	0	816636885	InvalidRead	nxt-bad	Not Yet	catchconv	Download?	premade@metafuzz.com
980901	0	816637091	InvalidRead	nxt-bad	Not Yet	catchconv	Download?	premade@metafuzz.com
496868	0	1495414164	Leak_DefinitelyLost	samate	Not Yet	catchconv	Download?	premade@metafuzz.com
496868	0	4267069708	InvalidWrite	samate	Not Yet	catchconv	Download?	premade@metafuzz.com

SmartFuzz and zzuf Find Different Bugs (Under Fuzzy Stack Hash)



Both SmartFuzz and zzuf Find Bugs in Five out of Six Test Programs

	mplayer		ffmpeg		exiv2		gzip		convert	
SyscallParam	4	3	2	3	0	0	0	0	0	0
UninitCondition	13	1	1	8	0	0	0	0	3	8
UninitValue	0	3	0	3	0	0	0	0	0	2
Overlap	0	0	0	1	0	0	0	0	0	0
Leak_DefinitelyLost	2	2	2	4	0	0	0	0	0	0
Leak_PossiblyLost	2	1	0	2	0	1	0	0	0	0
InvalidRead	1	2	0	4	4	6	2	0	1	1
InvalidWrite	0	1	0	3	0	0	0	0	0	0
Total	22	13	5	28	4	7	2	0	4	11
Cost per bug	$1.30	$2.16	$5.76	$1.03	$1.80	$1.03	$3.60	NA	$1.20	$0.65

No bugs found in bzip2.
Cost per bug at Amazon EC2 rates February 2009.

SmartFuzz Beats zzuf on Two Out of Six Test Programs

	mplayer		ffmpeg		exiv2		gzip		convert	
SyscallParam	4	3	2	3	0	0	0	0	0	0
UninitCondition	13	1	1	8	0	0	0	0	3	8
UninitValue	0	3	0	3	0	0	0	0	0	2
Overlap	0	0	0	1	0	0	0	0	0	0
Leak_DefinitelyLost	2	2	2	4	0	0	0	0	0	0
Leak_PossiblyLost	2	1	0	2	0	1	0	0	0	0
InvalidRead	1	2	0	4	4	6	2	0	1	1
InvalidWrite	0	1	0	3	0	0	0	0	0	0
Total	22	13	5	28	4	7	2	0	4	11
Cost per bug	$1.30	$2.16	$5.76	$1.03	$1.80	$1.03	$3.60	NA	$1.20	$0.65

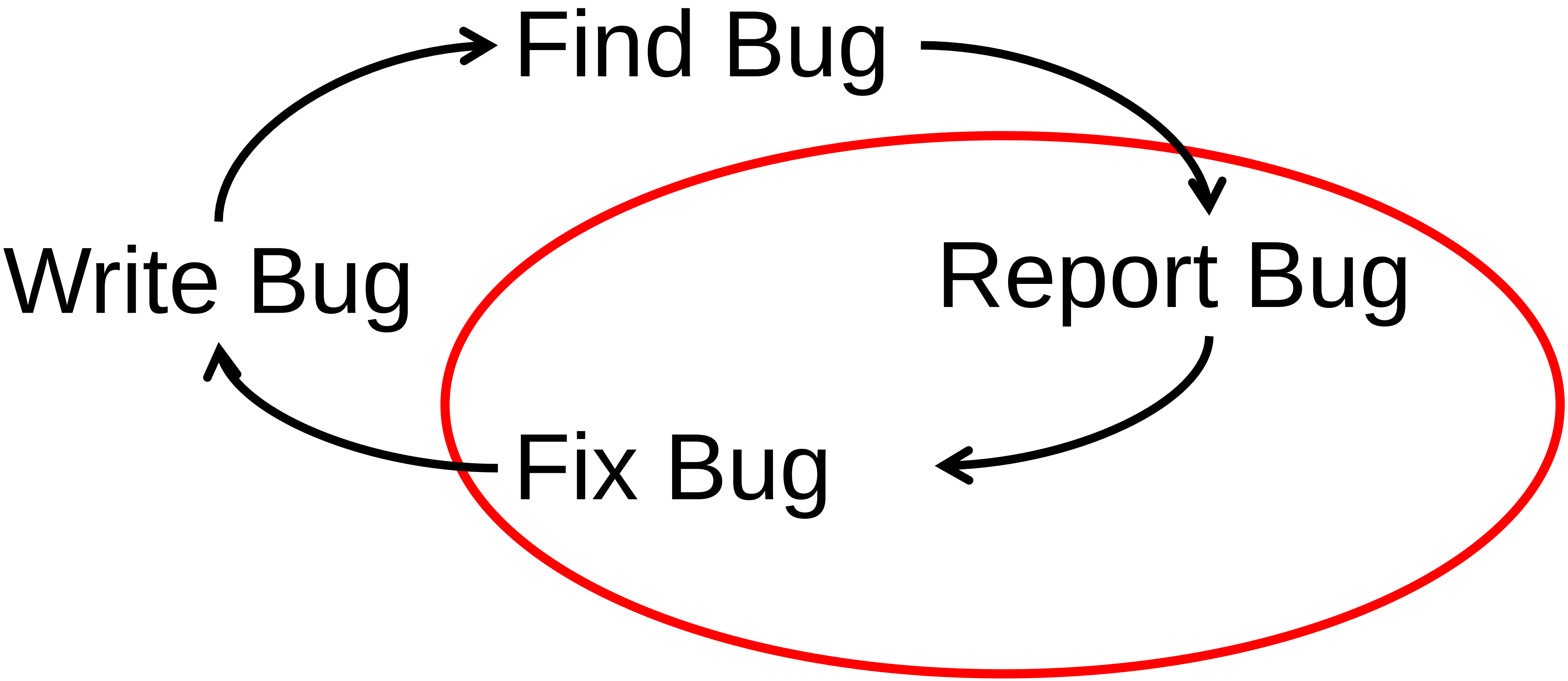
Bug-Seeking Queries Yield Bugs

	Queries	Test Cases	Bugs
Coverage	588068	31121	19
ConversionNot32	4586	0	0
Conversion32to8	1915	1377	3
Conversion32to16	16073	67	4
Conversion16Sto32	206	0	0
SignedOverflow	167110	0	0
SignedUnderflow	20198	21	3
UnsignedOverflow	164155	9280	3
MallocArg	30	0	0
SignedUnsigned	125509	6949	5

Theory Plus Practice

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

What Next?



Thank you!

Questions?

dmolnar@eecs.berkeley.edu