

# Two Techniques for Automatically Eliminating Software Defects

Martin Rinard

MIT EECS, MIT CSAIL

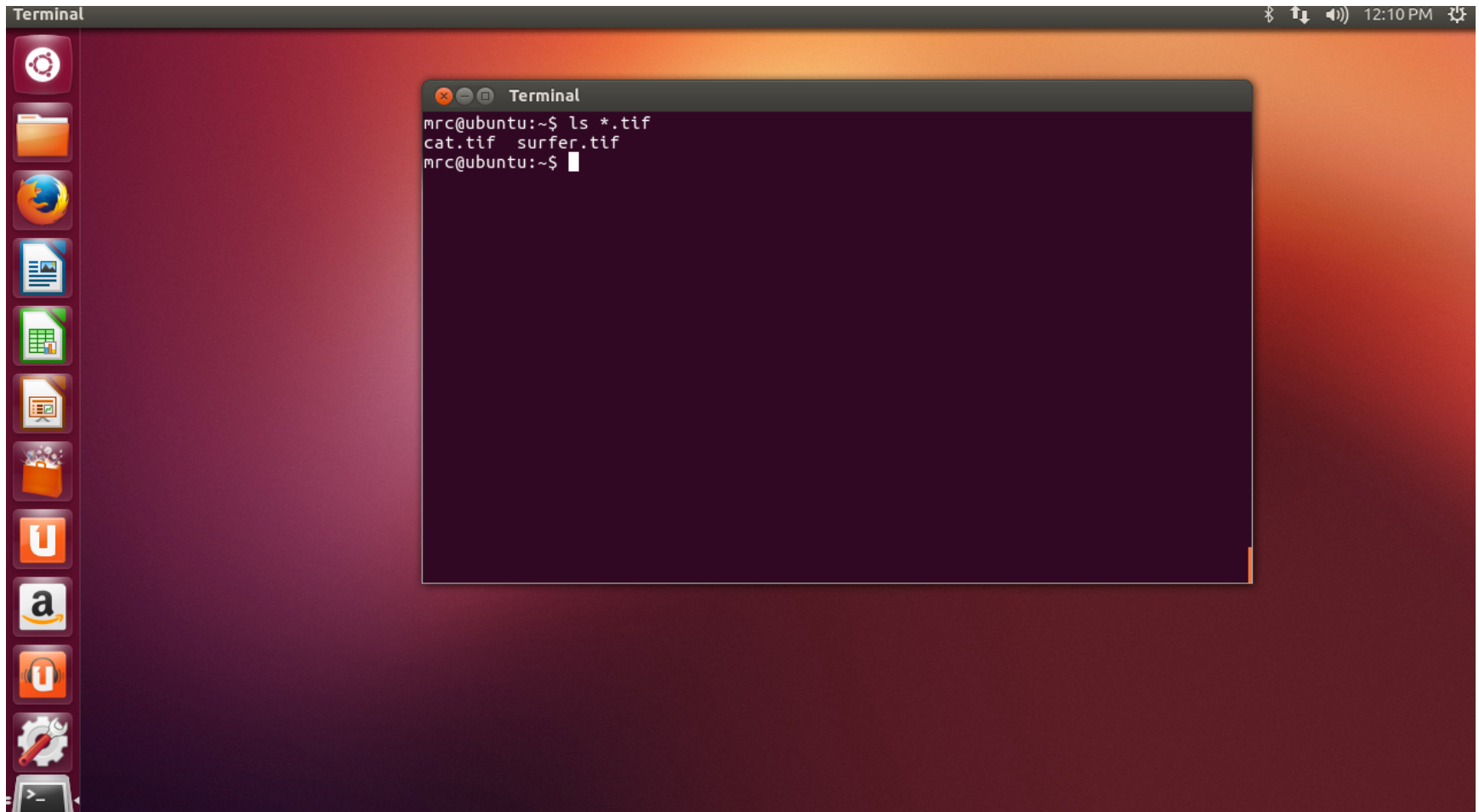
Massachusetts Institute of Technology

Cambridge, MA 02139

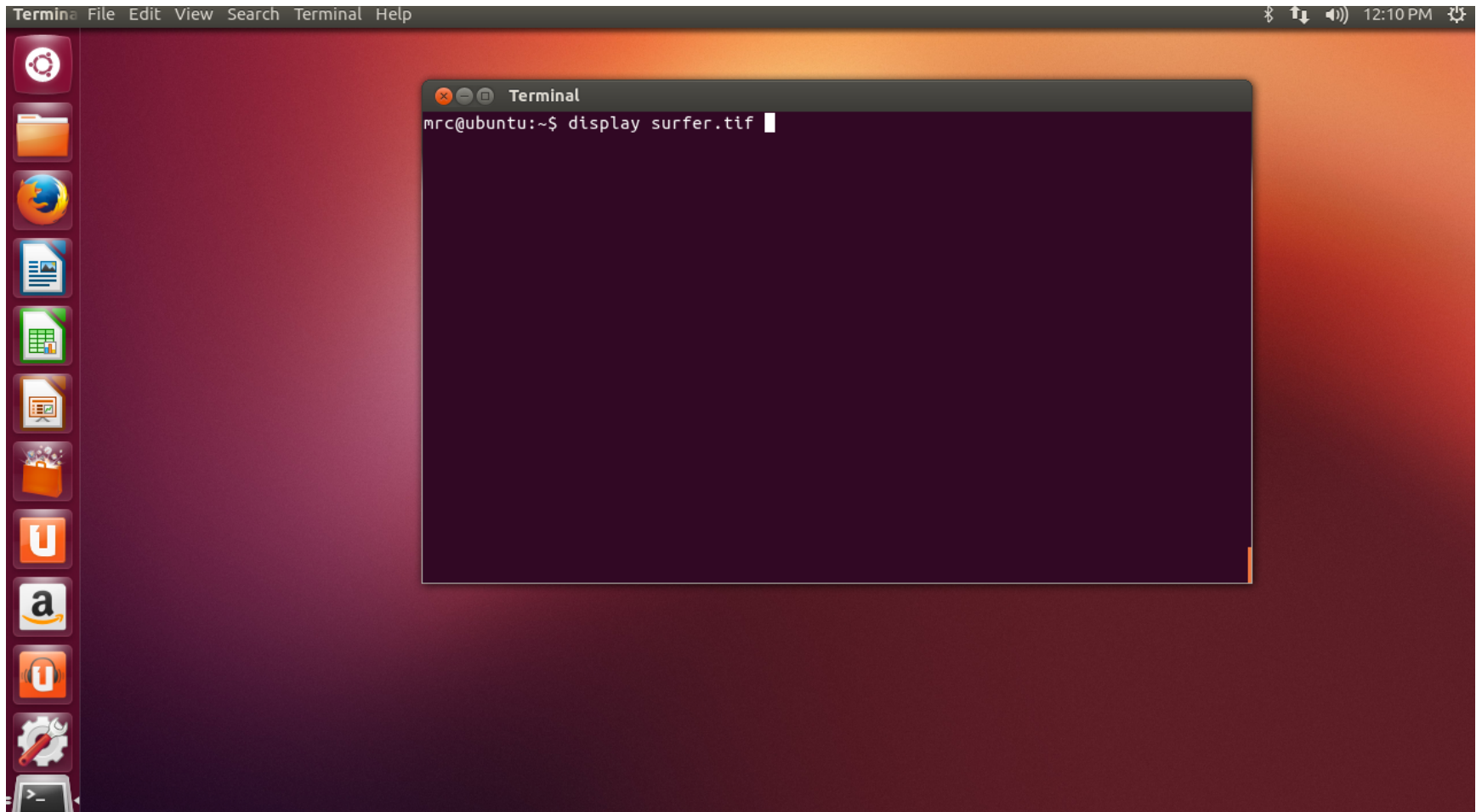
# ImageMagick Display

- Popular, free and open-source software suite
  - Displaying, converting and editing images
  - Read/Write > 200 formats
- Very popular for users && programs
  - Drupal, MediaWiki, phpBB, Vbulletin, etc.,

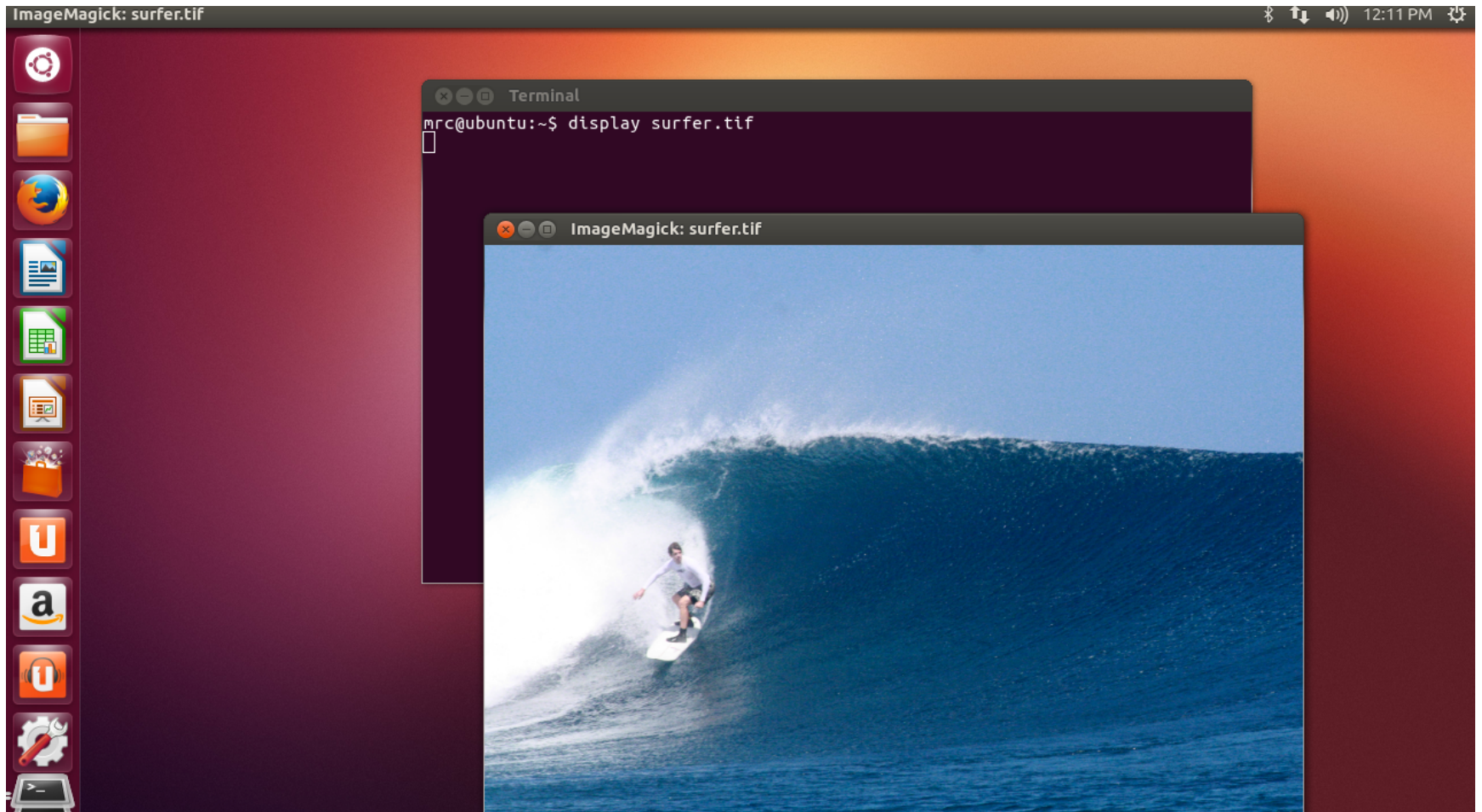
# Images



# Display surfer

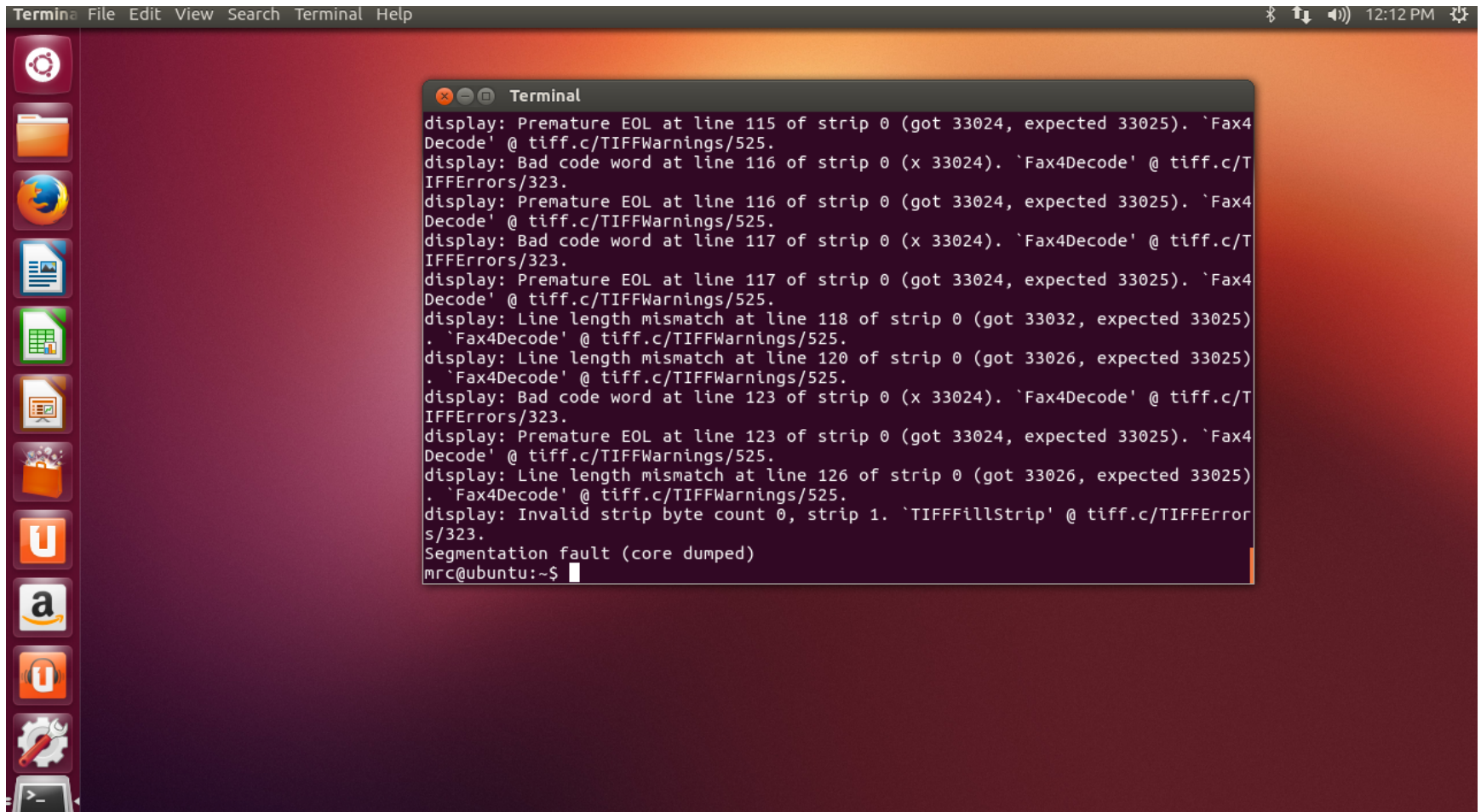


# Show Surfer



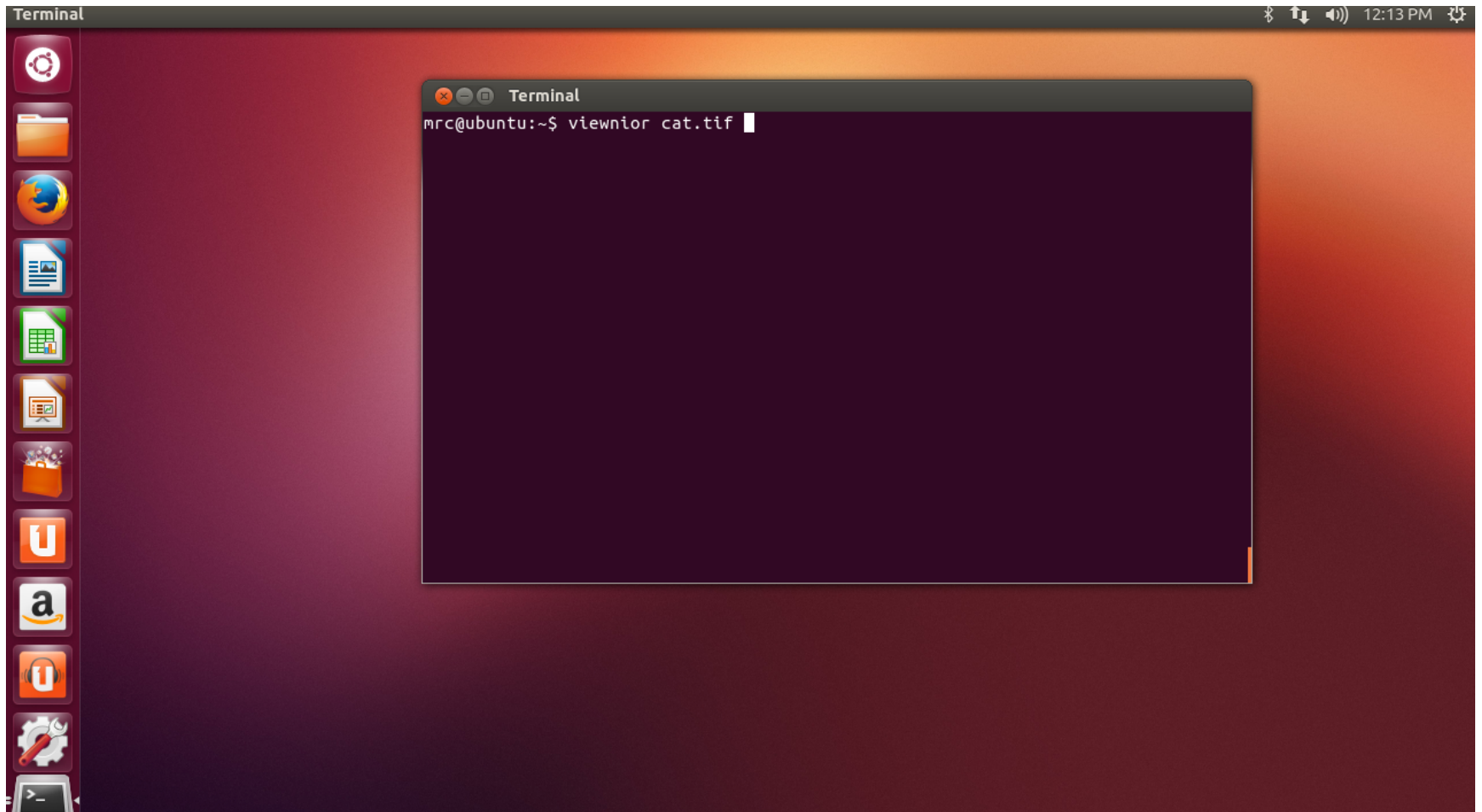


# Cat Crash



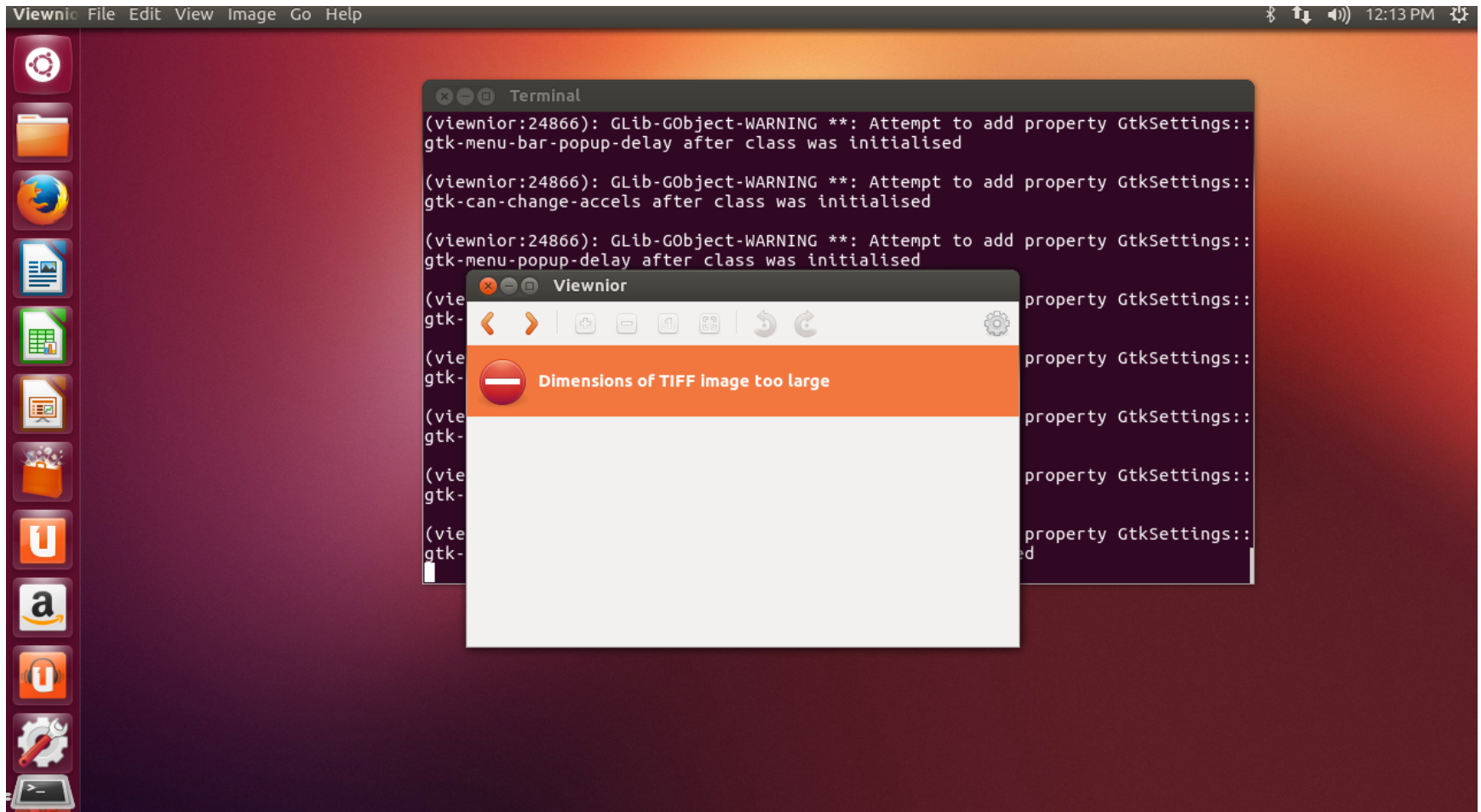
```
Terminal File Edit View Search Terminal Help 12:12 PM
Terminal
display: Premature EOL at line 115 of strip 0 (got 33024, expected 33025). `Fax4
Decode' @ tiff.c/TIFFWarnings/525.
display: Bad code word at line 116 of strip 0 (x 33024). `Fax4Decode' @ tiff.c/T
IFFErrors/323.
display: Premature EOL at line 116 of strip 0 (got 33024, expected 33025). `Fax4
Decode' @ tiff.c/TIFFWarnings/525.
display: Bad code word at line 117 of strip 0 (x 33024). `Fax4Decode' @ tiff.c/T
IFFErrors/323.
display: Premature EOL at line 117 of strip 0 (got 33024, expected 33025). `Fax4
Decode' @ tiff.c/TIFFWarnings/525.
display: Line length mismatch at line 118 of strip 0 (got 33032, expected 33025)
. `Fax4Decode' @ tiff.c/TIFFWarnings/525.
display: Line length mismatch at line 120 of strip 0 (got 33026, expected 33025)
. `Fax4Decode' @ tiff.c/TIFFWarnings/525.
display: Bad code word at line 123 of strip 0 (x 33024). `Fax4Decode' @ tiff.c/T
IFFErrors/323.
display: Premature EOL at line 123 of strip 0 (got 33024, expected 33025). `Fax4
Decode' @ tiff.c/TIFFWarnings/525.
display: Line length mismatch at line 126 of strip 0 (got 33026, expected 33025)
. `Fax4Decode' @ tiff.c/TIFFWarnings/525.
display: Invalid strip byte count 0, strip 1. `TIFFFillStrip' @ tiff.c/TIFFError
s/323.
Segmentation fault (core dumped)
mrc@ubuntu:~$
```

# ViewNior Cat

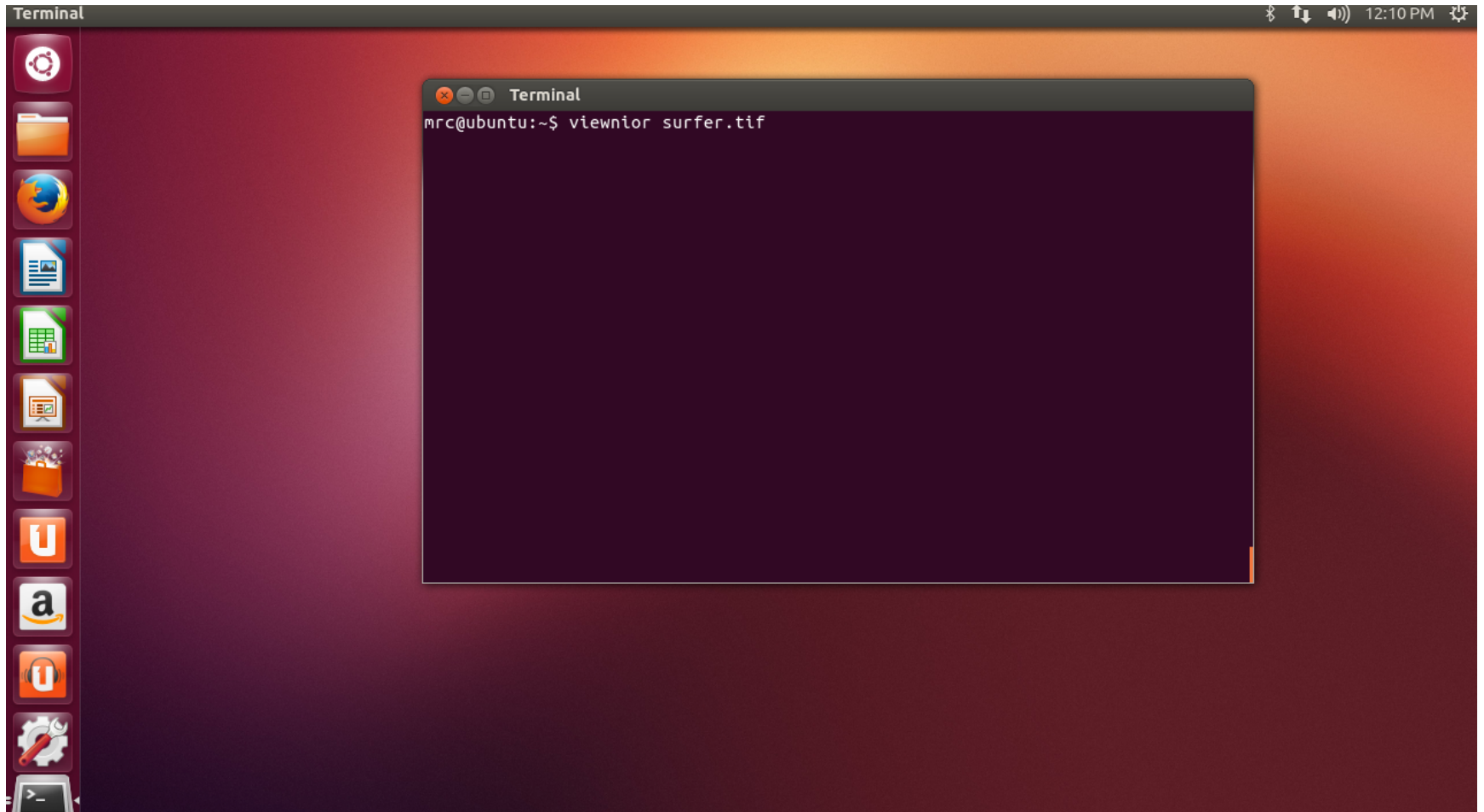




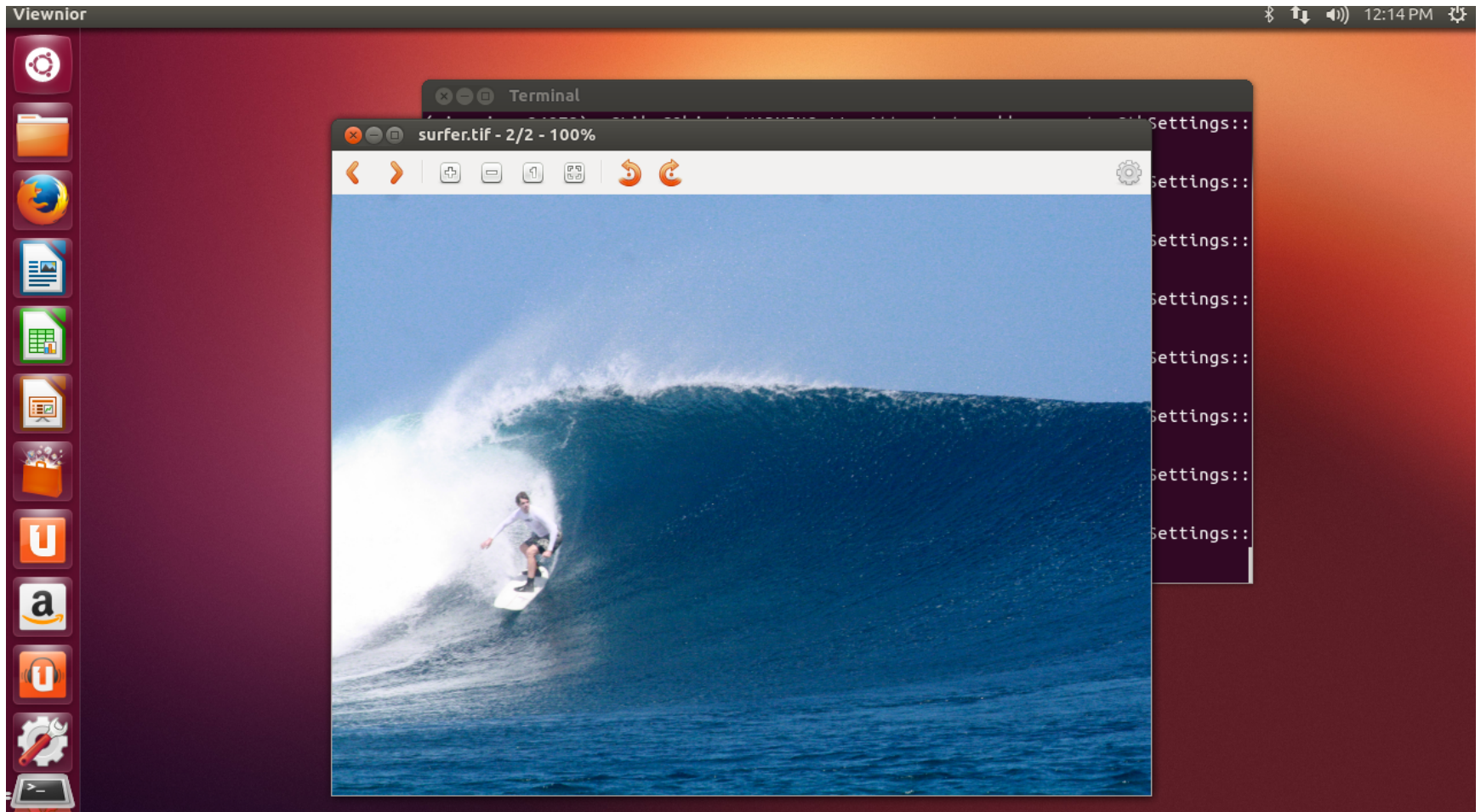
# ViewNior Protection



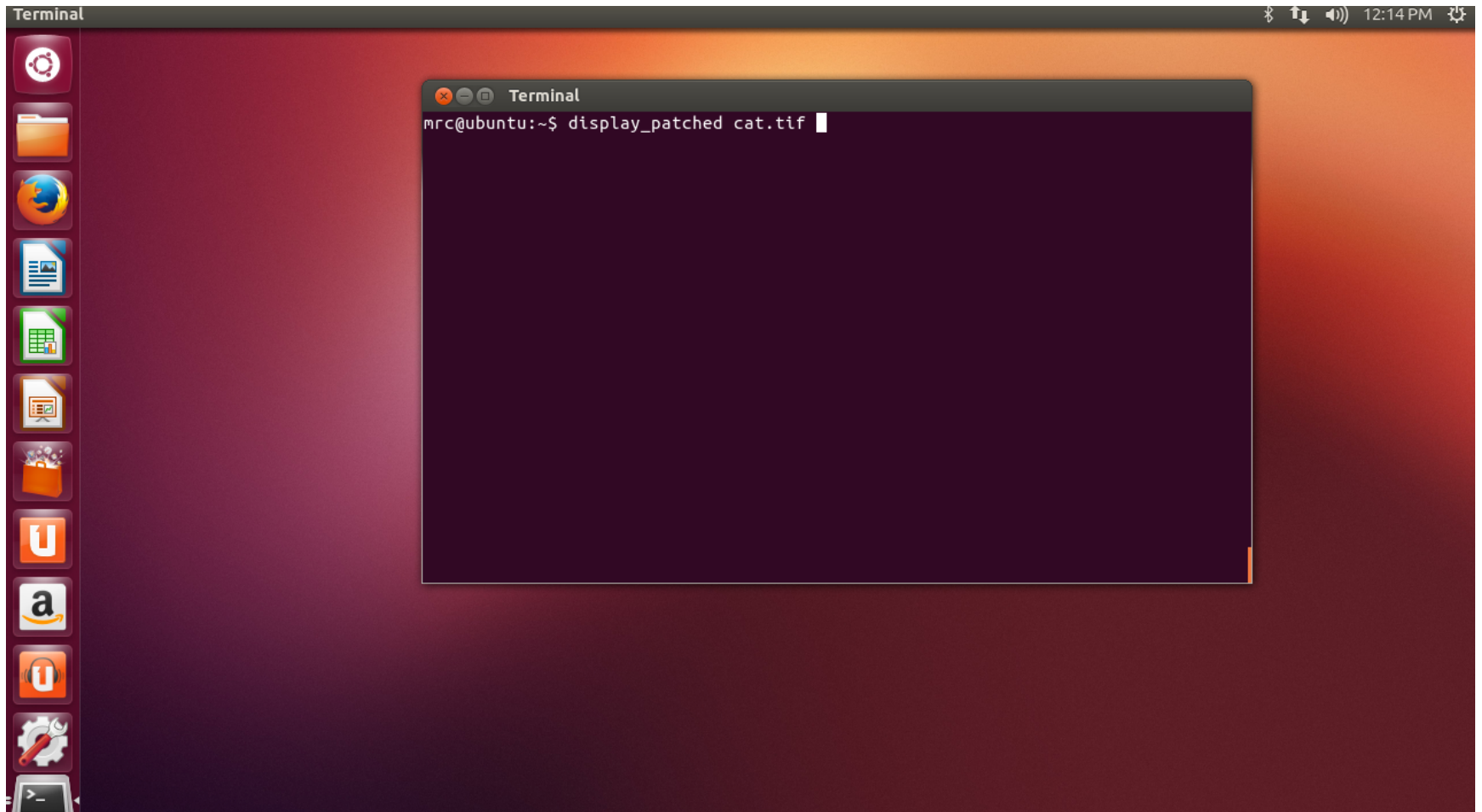
# ViewNior Surfer



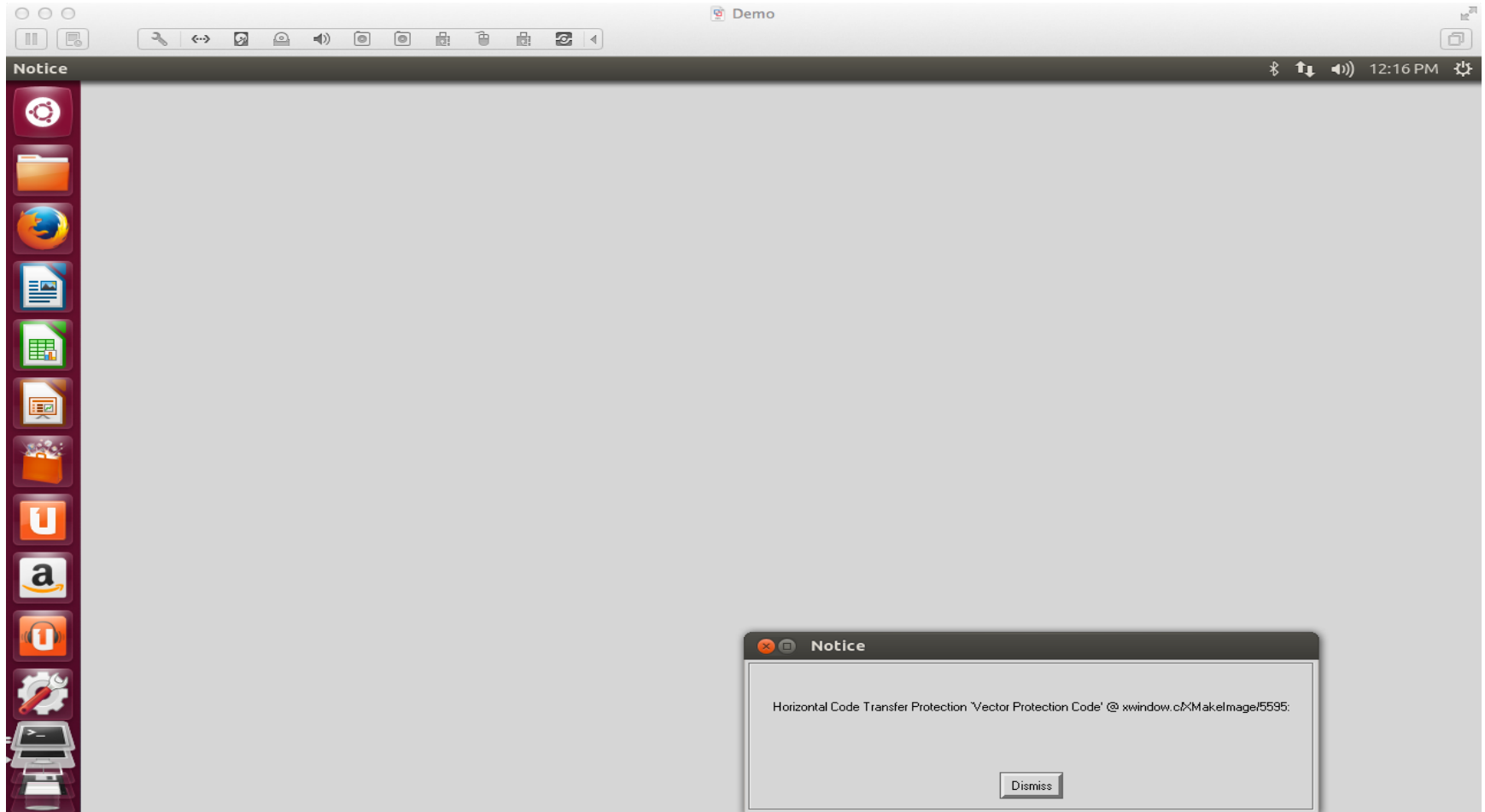
# ViewNior Surfer OK



# Patch Display



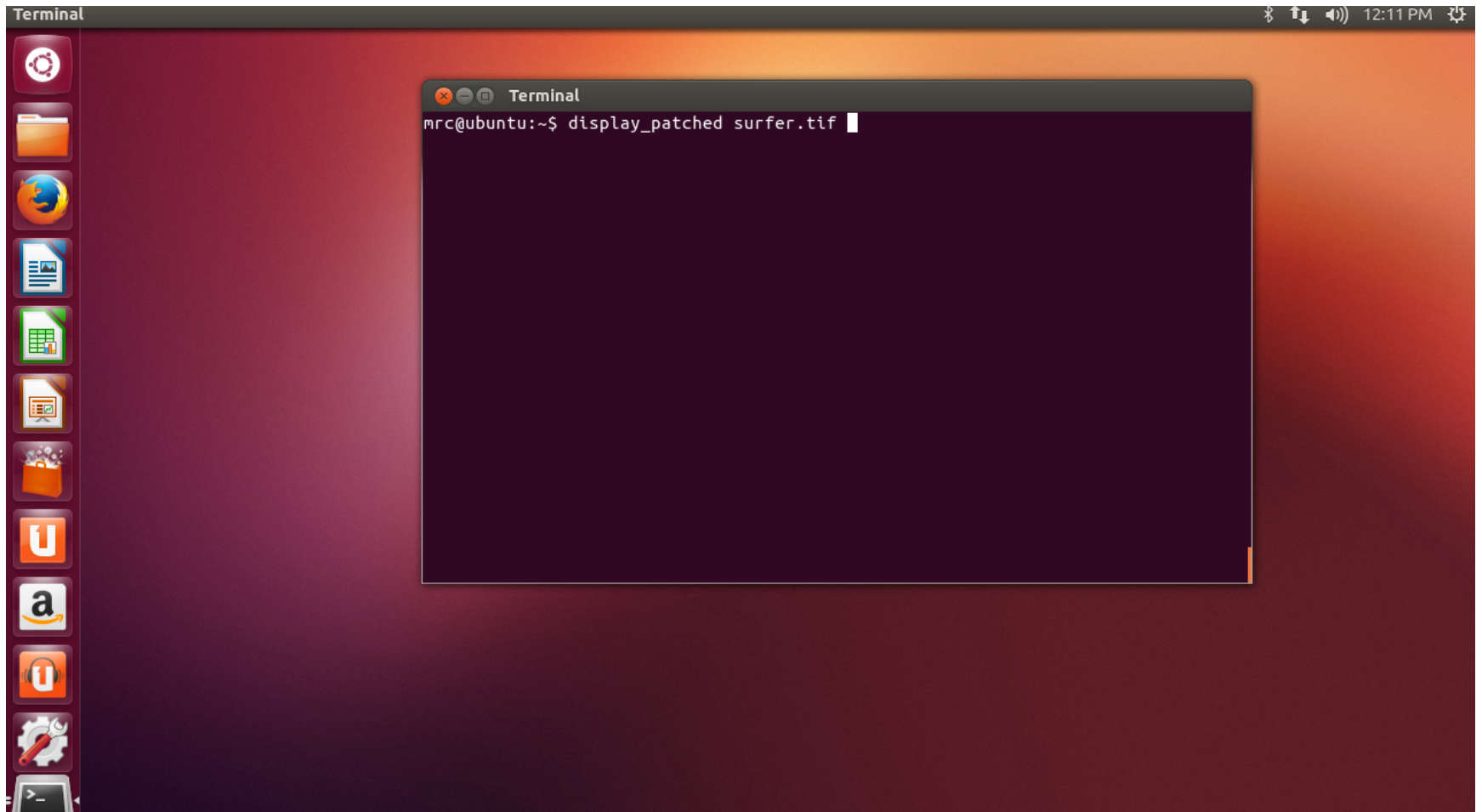
# Patched Display protects Cat



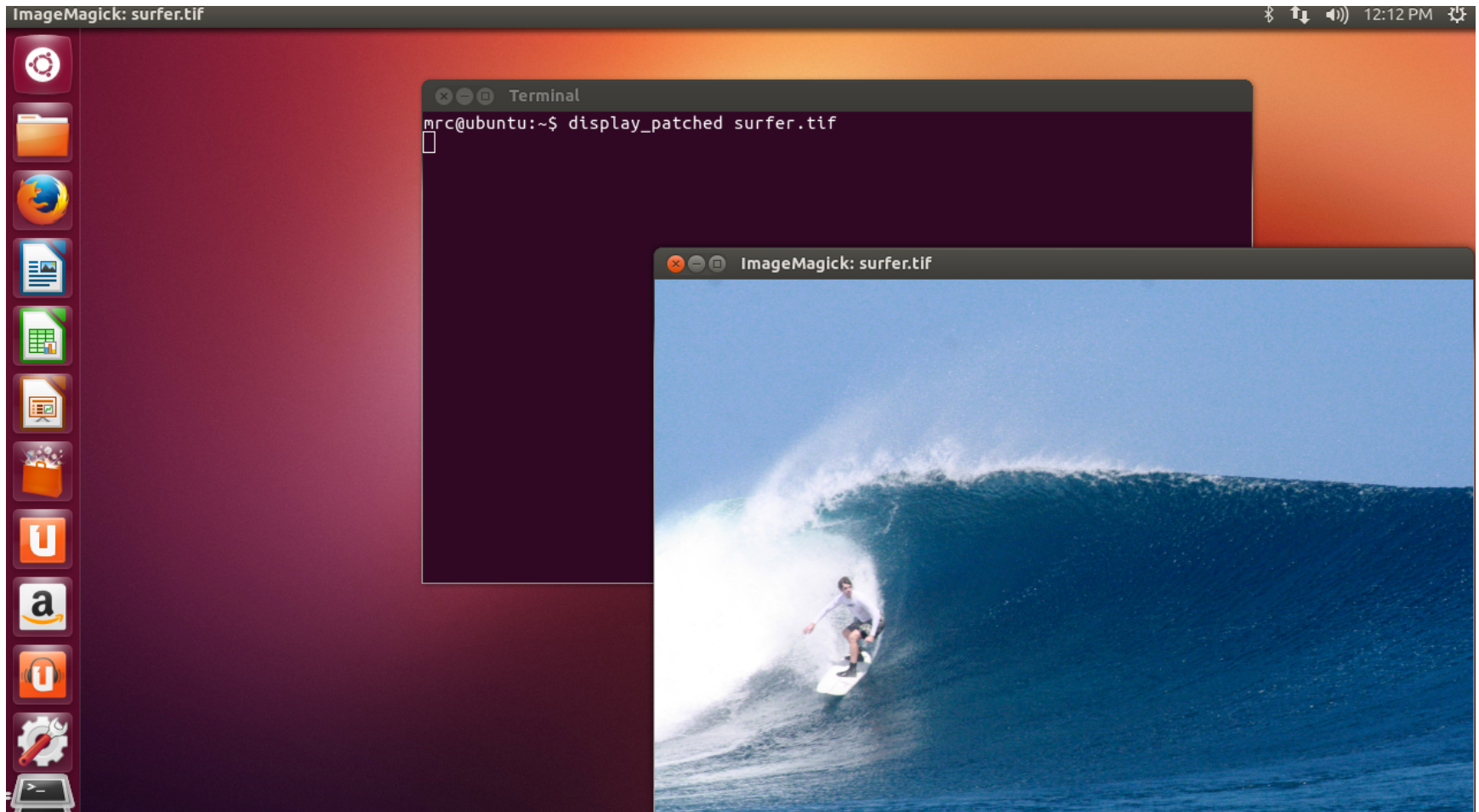


```
Terminal
mrc@ubuntu:~$ display_patched surfer.tif
mrc@ubuntu:~$ display_patched cat.tif
Horizontal Code Transfer reject input
mrc@ubuntu:~$
```

# Display patched surfer



# Display patched surfer

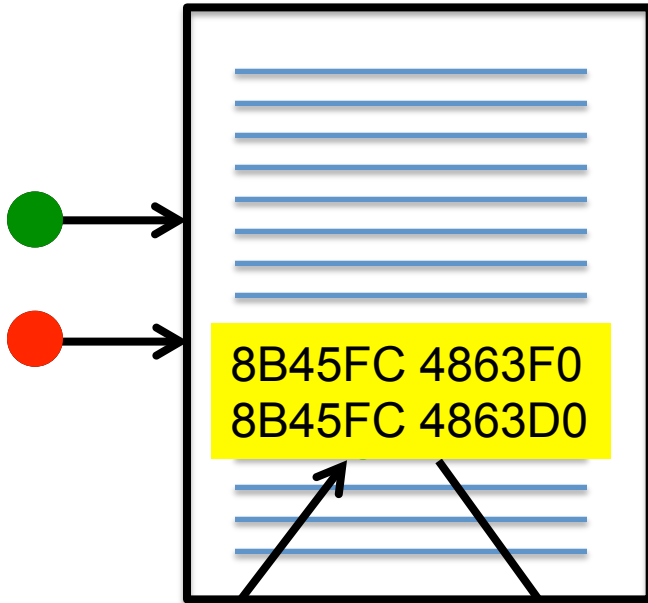




# CodePhage (CP) Overview

Donor

viewnior 1.4  
(stripped binary)



1. Locate Check

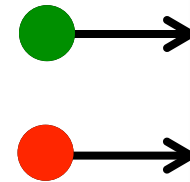
2. Extract Check

Application-Independent Representation of Check

Recipient

display 6.5.2  
(source code)

3. Identify Patch  
Insertion Point

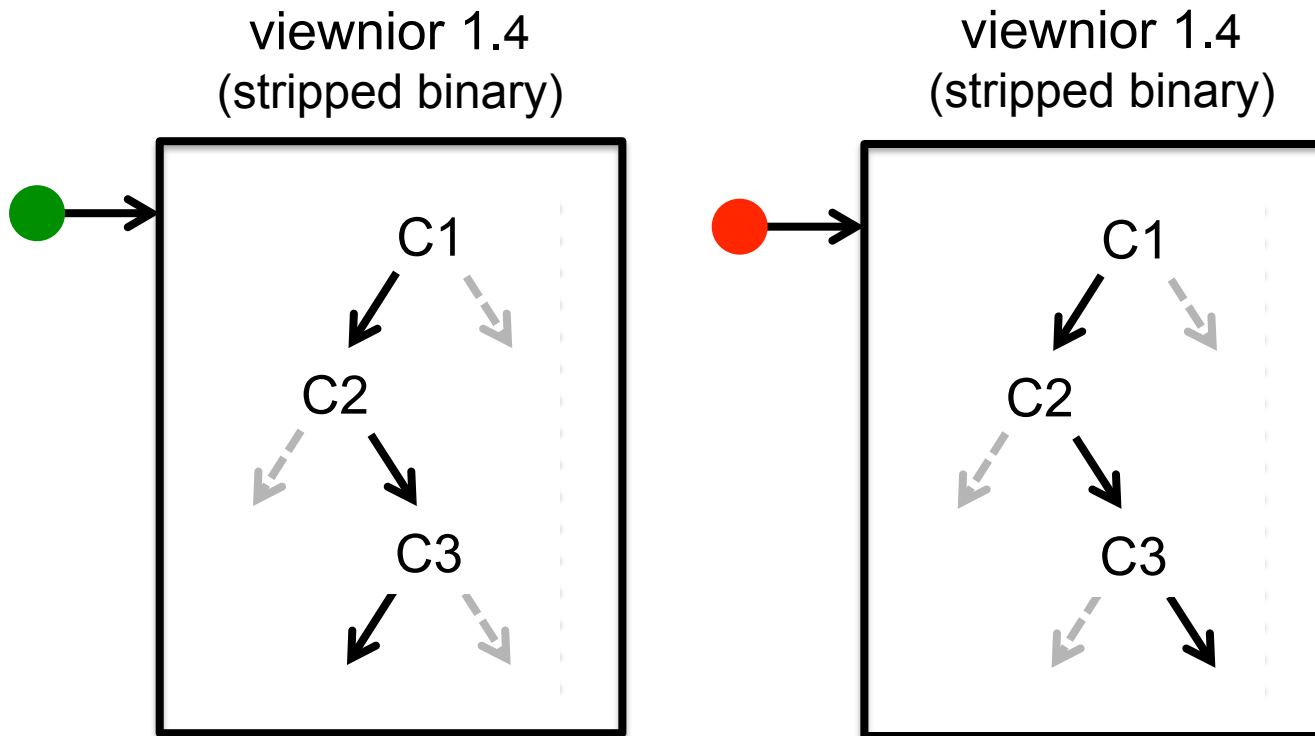


5. Verify Patch

4. Translate and Insert  
(source code patch)

# Locate Check

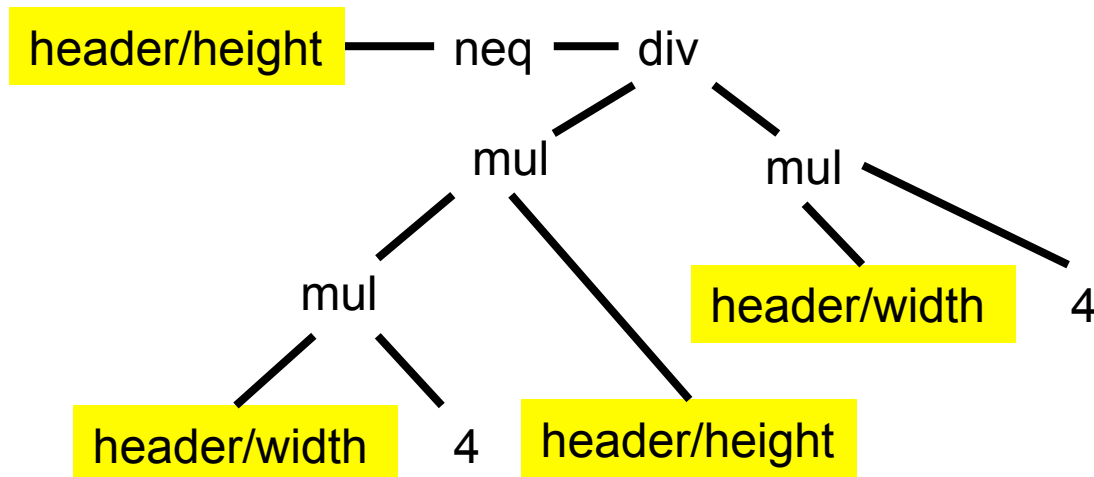
- Execute instrumented version of donor (viewnior 1.4)
- Record trace of executed conditional branches
- Find flipped branches on malicious input



Hypothesis:  
C3 is the  
missing  
check in  
recipient

# Extract Check

- Start with a conditional branch instruction  
**jne label, je label, jle label, ...**
- Goal: Obtain application-independent check
- Symbolic expression tree for condition
  - Internal nodes are operations (add, sub, cmp, ...)
  - Leaves are constants and input fields



Integer  
Overflow  
Check

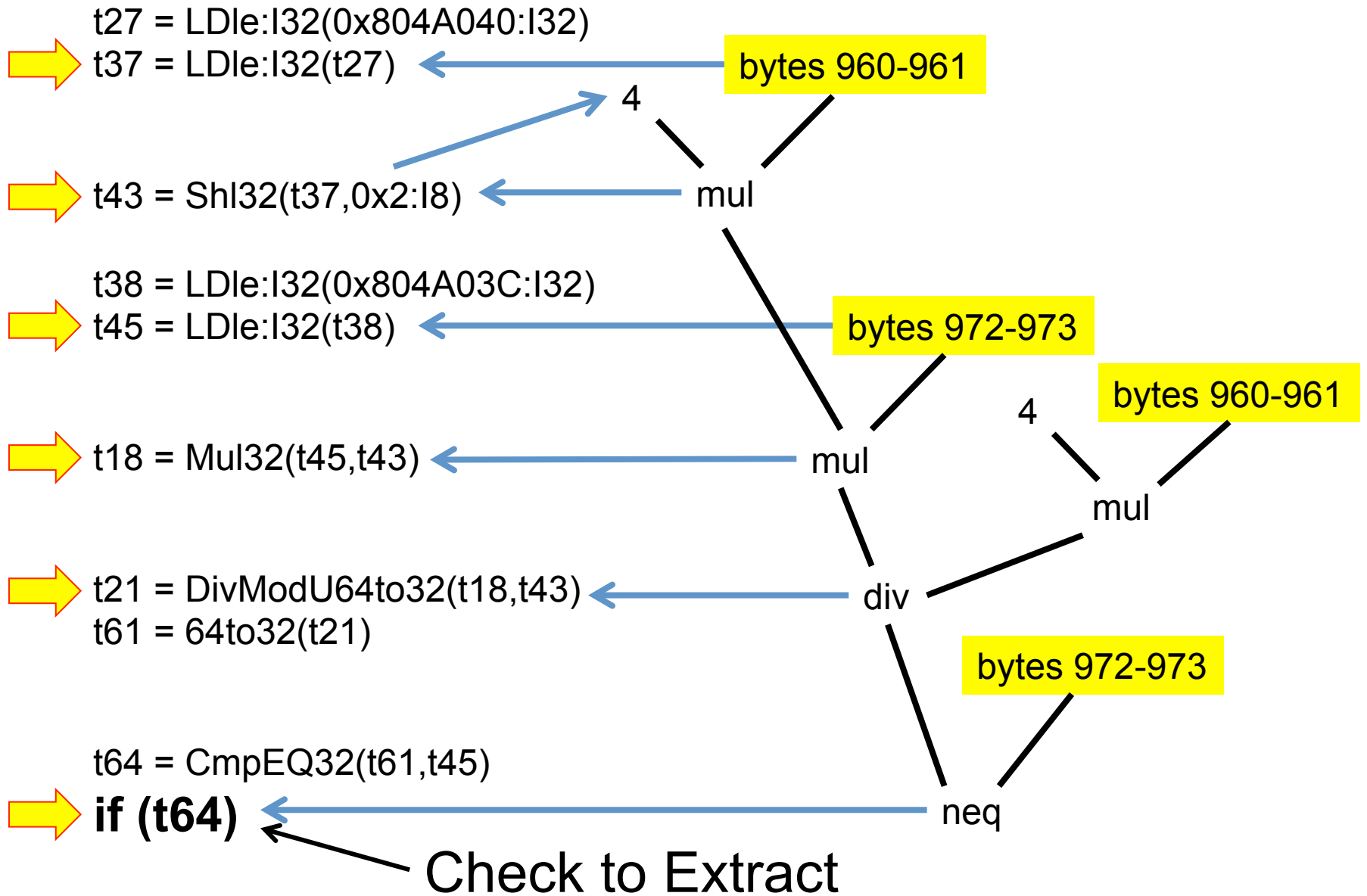
# Challenges

- Condition can be computed by arbitrarily complex sequence of binary instructions
  - Need to extract computed logical expression
  - Can involve arbitrary computations throughout application
- Input bytes can flow all over address space
  - Condition typically references input bytes (or derived values) as stored in application-specific data structures
  - Working with compiled data structures encoded in flat address space of stripped binary
- Application works with raw bytes from input file
- But symbolic expression tree uses symbolic input fields

# Symbolic Expression Tracing

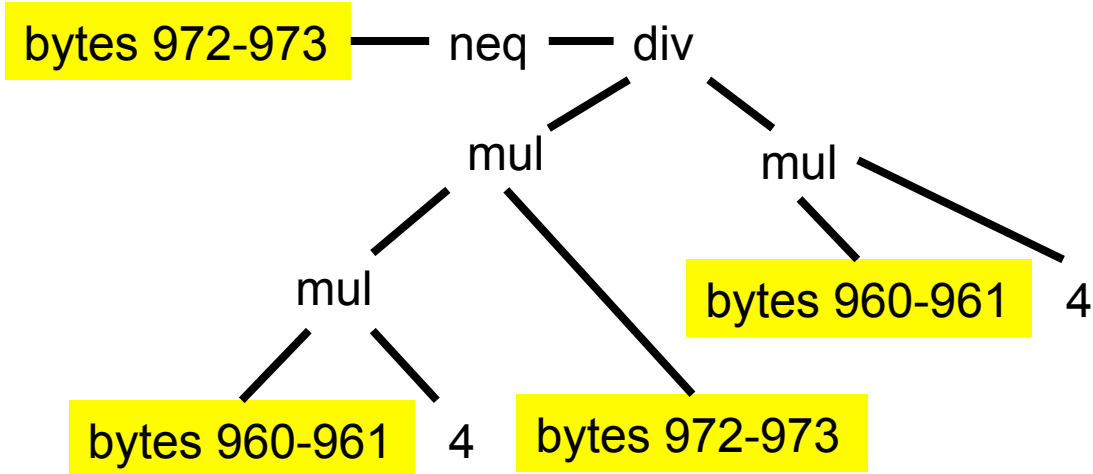
- Goal – symbolic expression tree for condition
  - Internal nodes are operations (add, sub, cmp, ...)
  - Leaves are constants and input bytes
- Execute instrumented application (Valgrind)
- Record expression derivation information
  - Instrument system call I/O operations  
(record mapping between input bytes and memory)
  - Trace VEX IR operations (add, sub, cmp, mv, ...)  
(record how result derived from operands)

# VEX IR From Donor

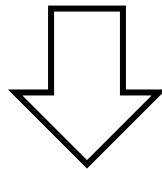


# Is This Plausible?

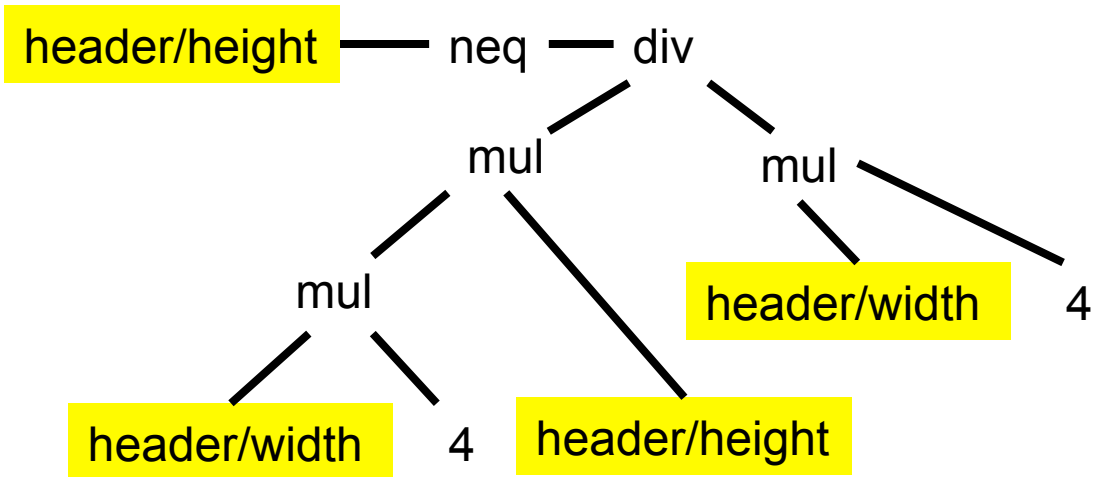
- Maybe, but we don't do it - too much overhead
- So we use an optimization – two executions
  - First execution: value tracing
    - Record bytes that affect each computed value
    - Determine *relevant bytes*  
(input bytes that influence condition)
  - Second execution: symbolic expression tracing
  - But only for values that involve relevant bytes



Hachoir



From Byte Offsets To Symbolic Input Fields





# Extracted Condition from viewnior

```
Conjunction(ULessEqual(32,Add(32,Add(32,Mul(32,Add(32,BvOr(32,Constant(0x00),ToSize(32,UShr(32,BvAnd(32,HachField(32, '/
header/
width'),Constant(0xFF000000)),Constant(24))))),Add(32,Add(32,BvOr(32,Constant(0x00),Shl(32,ToSize(32,BvAnd(32,HachField(32, '/
header/width'),Constant(0xFF))),Constant(24))),BvOr(32,Constant(0x00),Shl(32,ToSize(32,UShr(32,BvAnd(32,HachField(32, '/header/
width'),Constant(0xFF00)),Constant(8))),Constant(16))))),BvOr(32,Constant(0x00),Shl(32,ToSize(32,UShr(32,BvAnd(32,HachField(32, '/
header/
width'),Constant(0xFF0000
```

```
nd(32,HachField(32, '/heac
height'),Constant(0xFF000
header/height'),Constant(0
height'),Constant(0xFF00),
header/
height'),Constant(0xFF000
Siz(32,UShr(32,BvAnd(32,
height'),Constant(0xFF000
header/height'),Constant(0
height'),Constant(0xFF00),
header/
height'),Constant(0xFF000
, '/header/
width'),Constant(0xFF000C
header/width'),Constant(0x
width'),Constant(0xFF00)),
header/width'),Constant(0x
ULessEqual(32,Shrink(32,|
height'),Constant(0xFF000
header/height'),Constant(0
height'),Constant(0xFF00),
header/
height'),Constant(0xFF000
, '/header/
```

```
width'),Constant(0xFF000000)),Constant(24))))),Add(32,Add(32,BvOr(32,Constant(0x00),Shl(32,ToSize(32,BvAnd(32,HachField(32, '/
header/width'),Constant(0xFF))),Constant(24))),BvOr(32,Constant(0x00),Shl(32,ToSize(32,UShr(32,BvAnd(32,HachField(32, '/header/
width'),Constant(0xFF00)),Constant(8))),Constant(16))))),BvOr(32,Constant(0x00),Shl(32,ToSize(32,UShr(32,BvAnd(32,HachField(32, '/
header/width'),Constant(0xFF0000)),Constant(16))),Constant(8))))),Constant(536870911))))
```

## Why?

### Big Endian to Little Endian

### Shifts and Masks

### Selection of Quotient from 64 Bit Divide

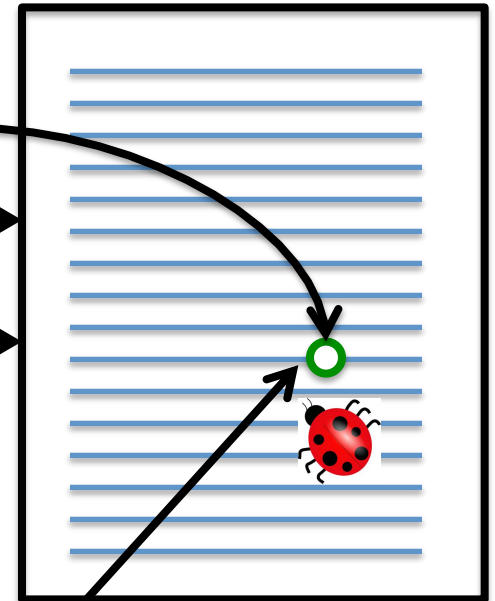
```
)0),ToSize(32,UShr(32,BvA
And(32,HachField(32, '/
,HachField(32, '/header/
vAnd(32,HachField(32, '/
vOr(32,Constant(0x00),To
And(32,HachField(32, '/
,HachField(32, '/header/
vAnd(32,HachField(32, '/
32,BvAnd(32,HachField(32
.nd(32,HachField(32, '/
HachField(32, '/header/
vAnd(32,HachField(32, '/
, '/header/
And(32,HachField(32, '/
,HachField(32, '/header/
vAnd(32,HachField(32, '/
32,BvAnd(32,HachField(32
```

# Identify Patch Insertion Point

Recipient

display 6.5.2  
(source code)

3. Identify Patch  
Insertion Point



Application-Independent Representation of Check

# Identify Patch Insertion Point in Recipient

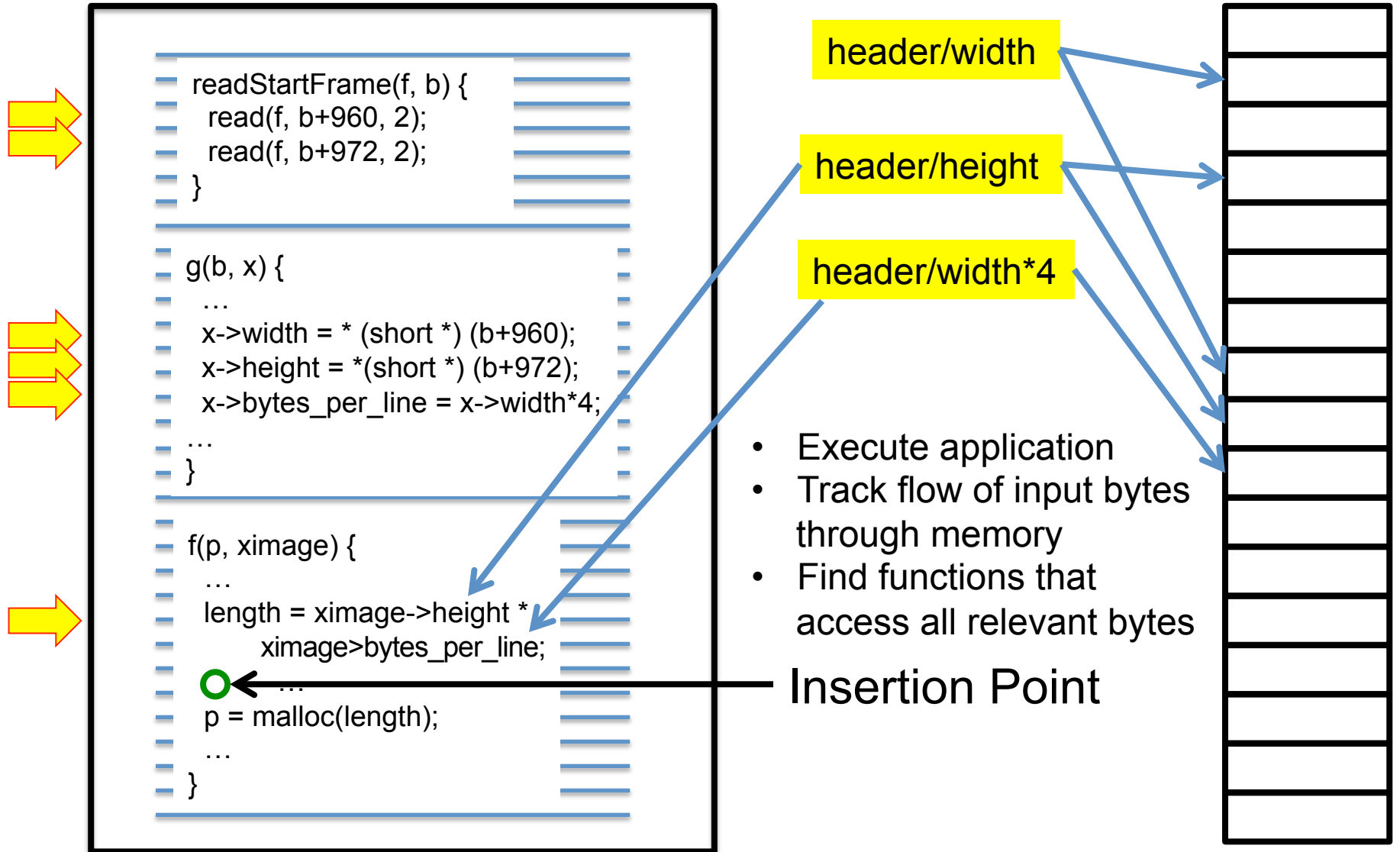
- Key issue: relevant input field values need to be available at patch insertion point
- Execute instrumented recipient
- Trace flow of values through application
  - Trace flow of input fields through memory/values
  - Find functions that access **all** relevant bytes (directly or indirectly via computed values)
  - Program points after last relevant load/store are potential patch insertion points

# Identify Patch Insertion Point

## Rationale

- If a function accesses relevant input bytes
- Then should be able to find source-level expressions in function for relevant bytes
- And so should be able to generate a source-level patch that uses relevant bytes

# Identify Patch Insertion Point



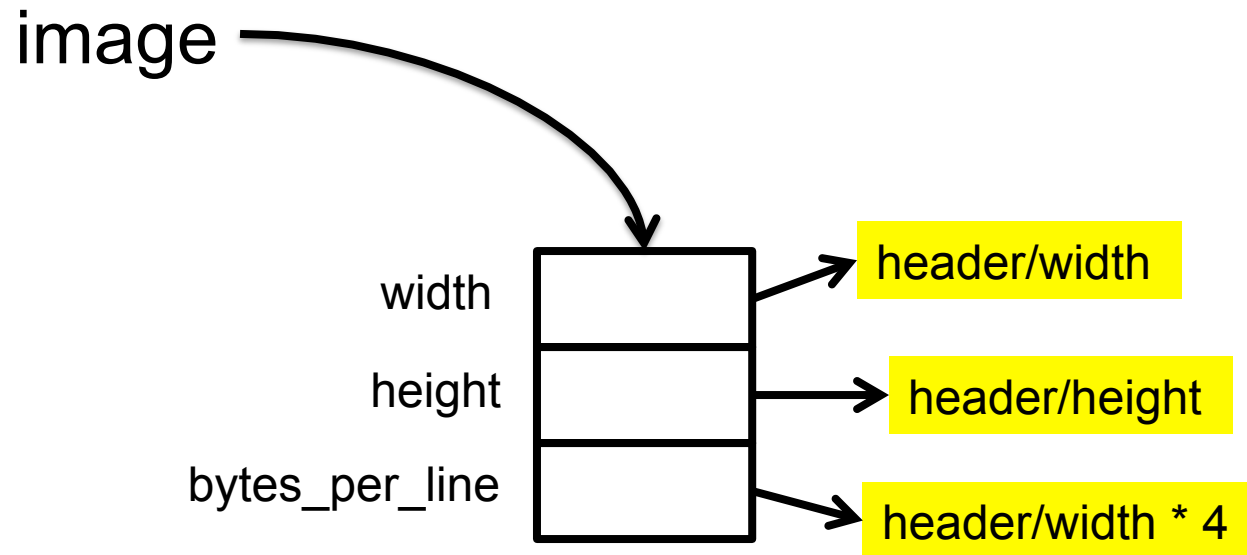
# Patch Translation

- Find source-level names for input bytes
- Step 1: Use debugging information to find roots
  - Local variables
  - Global variables
  - Parameters
- Step 2: Traverse from roots, find *available names*
  - Find values that involve relevant bytes
  - Record source-level expressions for those bytes
- Step 3: Use available names to translate patch into source code name space of recipient

# Finding Available Names

## Roots

length →  $(\text{header}/\text{width} * 4) * \text{header}/\text{height}$



# Available Names

## Available Names

## Values

length

$(\text{header}/\text{width} * 4) * \text{header}/\text{height}$

ximage->width

header/width

ximage->height

header/height

ximage->bytes\_per\_line

header/width \* 4



# Translate $E$ Into Source Code Name Space At Insertion Point

Application-independent  
representation of check  $E$

Set of source-code variables  
with input bytes  $Vars$

Directly translate constants

Use SMT solver to find  $V$   
with same value as  $E$

Otherwise decompose  
 $E$  and recurse

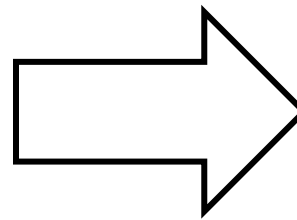
```
1  Parameters:
2   $E$ : A symbolic expression
3   $Vars$ : A set of active variables
4  For each  $V$  in  $Vars$ ,  $V.var$  is the variable
5  name;  $V.exp$  is the symbolic expression that
6  corresponds to the value of the variable.
7  Return:
8  Rewritten expression of  $E$  or
9  false if failed
10
11 Rewrite( $E$ ,  $Vars$ ) {
12   if ( $E$  is constant)
13     return  $E$ 
14   end if
15   for  $V$  in  $Vars$ 
16     if ( $SolverEquiv(E, V.exp)$ )
17        $Ret.opcode$   $\leftarrow$  VAR
18        $Ret.op1$   $\leftarrow$   $V.var$ 
19       return  $Ret$ ;
20     end if
21   end for
22   if ( $E.opcode$  is unary operation)
23      $Ret.opcode$   $\leftarrow$   $E.opcode$ 
24      $Ret.op1$   $\leftarrow$  Rewrite( $E.op1$ ,  $Vars$ )
25     if ( $Ret.op1$   $\neq$  false)
26       return  $Ret$ 
27     end if
28   else if ( $E.opcode$  is binary operation)
29      $Ret.opcode$   $\leftarrow$   $E.opcode$ 
30      $Ret.op1$   $\leftarrow$  Rewrite( $E.op1$ ,  $Vars$ )
31      $Ret.op2$   $\leftarrow$  Rewrite( $E.op2$ ,  $Vars$ )
32     if ( $Ret.op1$   $\neq$  false and
33          $Ret.op2$   $\neq$  false)
34       return  $Ret$ 
35     end if
36   end if
37   return false
38 }
```

# veiwnior to display Translation Example

*E*

```
BvOr(32, BvOr(32,
Constant(0x0), ToSize(32,
BvAnd(16, HachField(16, "/
ifd[0]/entry[0]/
value"), Constant(0xff))), Shl(32,
ToSize(32, UShr(32, BvAnd(16,
HachField(16, "/ifd[0]/entry[0]/
value"), Constant(0xff00)), Const
ant(0x8))), Constant(0x8)))
```

Rewrite(*E*, *Vars*)



ximage->height

# Final Patch

```
if (!((((unsigned int) (((unsigned long long) (((unsigned int) ((0 | ((unsigned long long) length)) |
    (((unsigned long long) (length >> ((unsigned int) 31))) << 32)))
    % ximage->bytes_per_line)) << 32) |
    ((unsigned long long) (((unsigned int) ((0 | ((unsigned long long) length)) |
    (((unsigned long long) (length >> ((unsigned int) 31))) << 32)))
    / ximage->bytes_per_line)))) == ximage->height))) {
printf("Horizontal Code Transfer reject input\n");
    ThrowXWindowFatalException(XServerError,
    "Horizontal Code Transfer Protection", "Test String");
exit(-1);
}
```

# Final Patch (simplified)

$(\text{header}/\text{width} * 4) * \text{header-height}$

$\text{header}/\text{height}$

$\text{header}/\text{width} * 4$

```
if (!((length / ximage->bytes_per_line) == ximage->height)) {  
    printf("Horizontal Code Transfer reject input\n");  
    ThrowXWindowFatalException(XServerError,  
        "Horizontal Code Transfer Protection", "Test String");  
    exit(-1);  
}
```

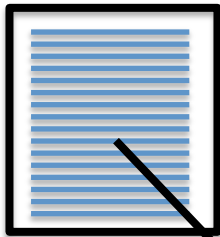
# Patch Journey

## viewnior source

```
rowstride = width * 4;  
bytes = height * rowstride;  
if (bytes / rowstride != height) {  
    /* overflow */  
}
```

gcc ↓

viewnior 1.4



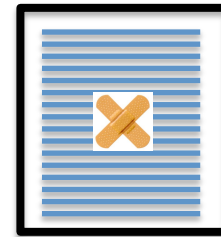
Find &  
Extract

## display source

```
if (!(length / ximage->bytes_per_line)  
    == ximage->height)) {  
    /* overflow */  
}
```

gcc ↓

Display 6.5.2



Translate &  
Insert

Application-Independent Representation of Check

# Variable Correspondence

viewnior  
source

```
rowstride = width * 4;  
bytes = height * rowstride;  
if (bytes / rowstride != height) {
```

display  
patch

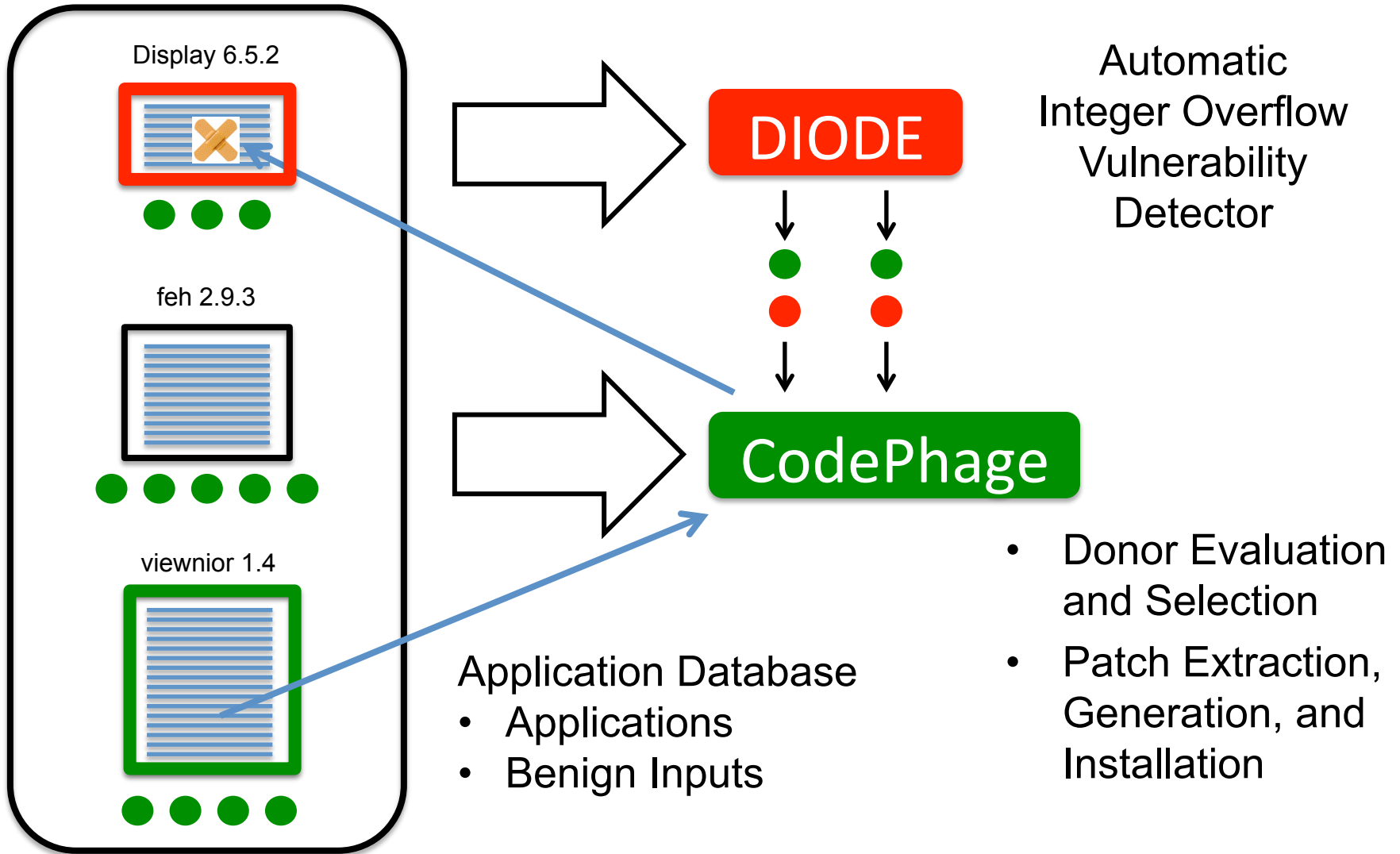
```
if (!(((length / ximage->bytes_per_line) == ximage->height)) {
```



# Patch Validation

- Run On Test Inputs
- For Integer Overflow Errors
  - Trace path to error
    - Record necessary conditions
    - Input fields in variables
  - Use SMT solver to find an input that
    - Satisfies necessary conditions
    - Causes integer overflow
  - If no such input exists, patch validates

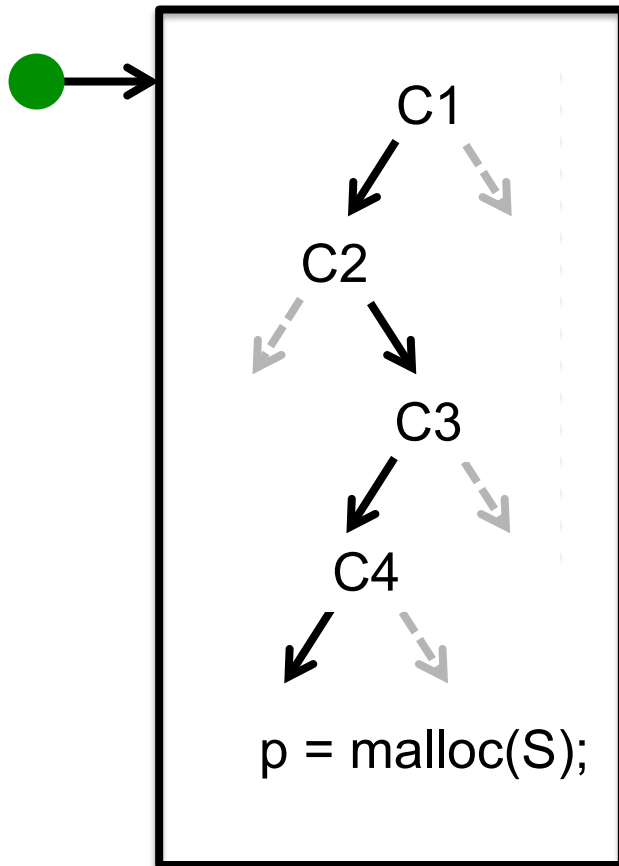
# Bigger Picture





# Finding Integer Overflows With DIODE

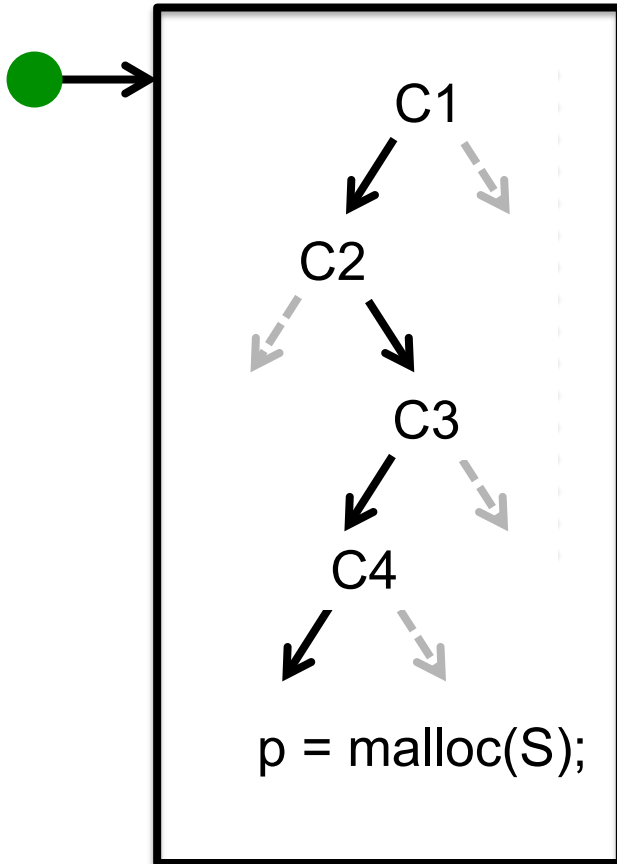
display 6.5.2



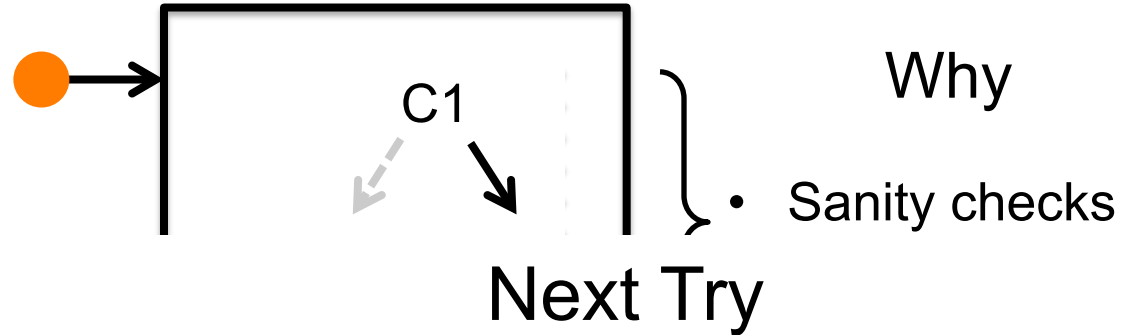
- Run (instrumented) application on benign input ●
- Encounter malloc(S) site
- Use solver to find new input ● that overflows S
- Run application on new input, see what happens
  - If reaches malloc(S) site
  - Guaranteed overflow!

# Common Outcome

display 6.5.2



display 6.5.2



Solver produces new input ● that

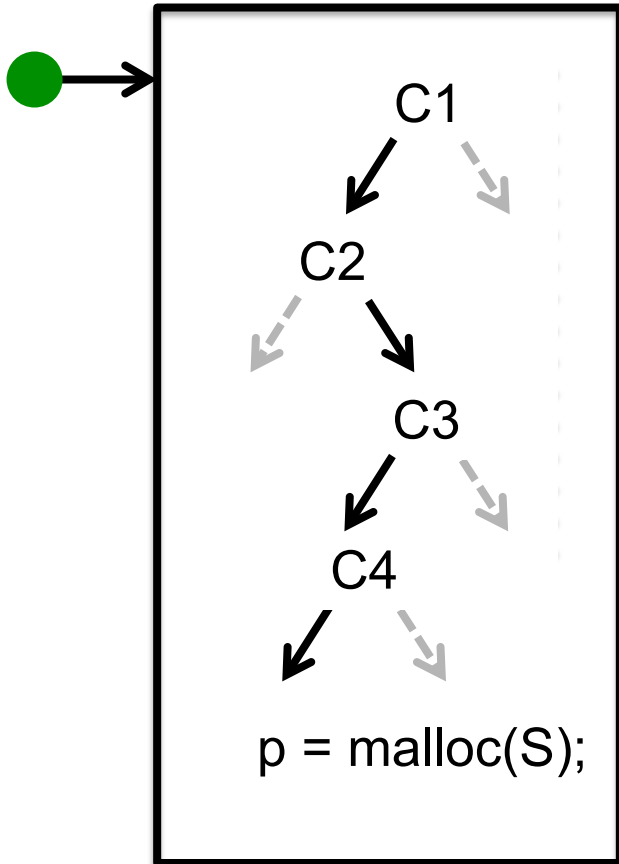
1. Overflows S, and
2. Follows same path as benign input ●

Common outcome

UNSAT - no such new input ● exists

# What DIODE Uses (Conceptually)

display 6.5.2

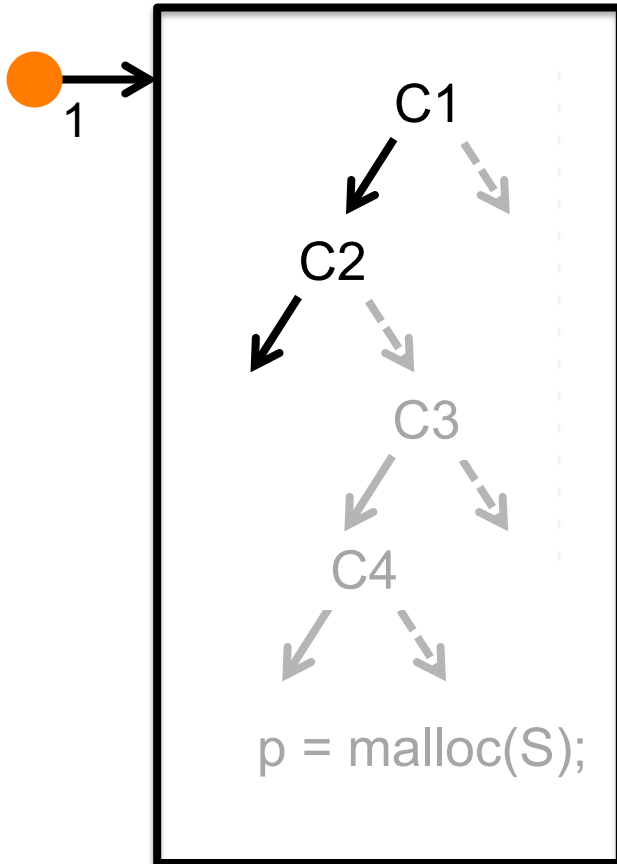


## Top Down Branch Enforcement

- Solver finds new input  $\bullet_1$  that
    1. Satisfies C1 and
    2. Overflows S
- If  $\bullet_1$  reaches malloc(S), integer overflow!

# What DIODE Uses (Conceptually)

display 6.5.2

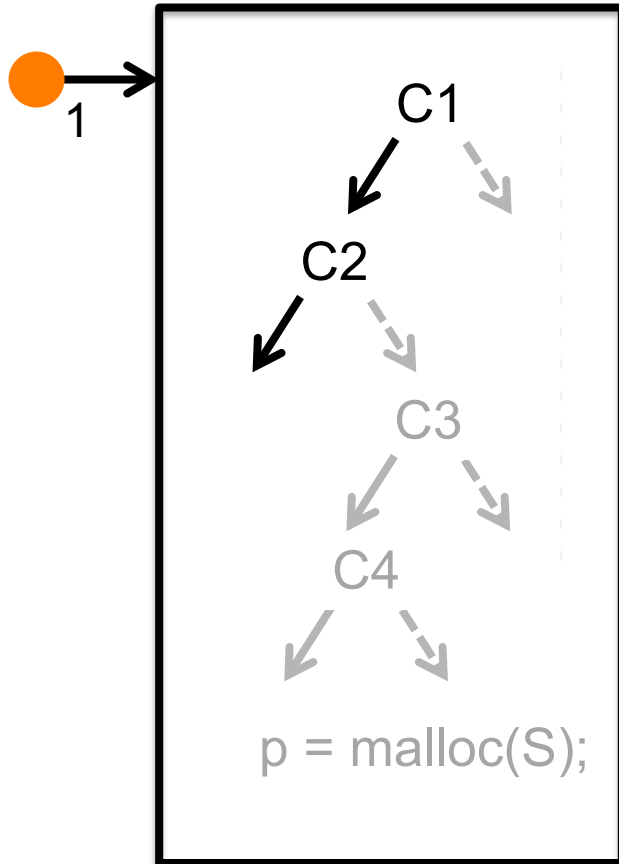


## Top Down Branch Enforcement

- Solver finds new input  $\bullet_1$  that
    1. Satisfies C1 and
    2. Overflows S
- If  $\bullet_1$  reaches malloc(S), integer overflow!

# What DIODE Uses (Conceptually)

display 6.5.2



## Top Down Branch Enforcement

- Solver finds new input ●<sub>1</sub> that

1. Satisfies C1 and
2. Overflows S

If ●<sub>1</sub> reaches malloc(S), integer overflow!

Otherwise solver finds new input ●<sub>2</sub> that

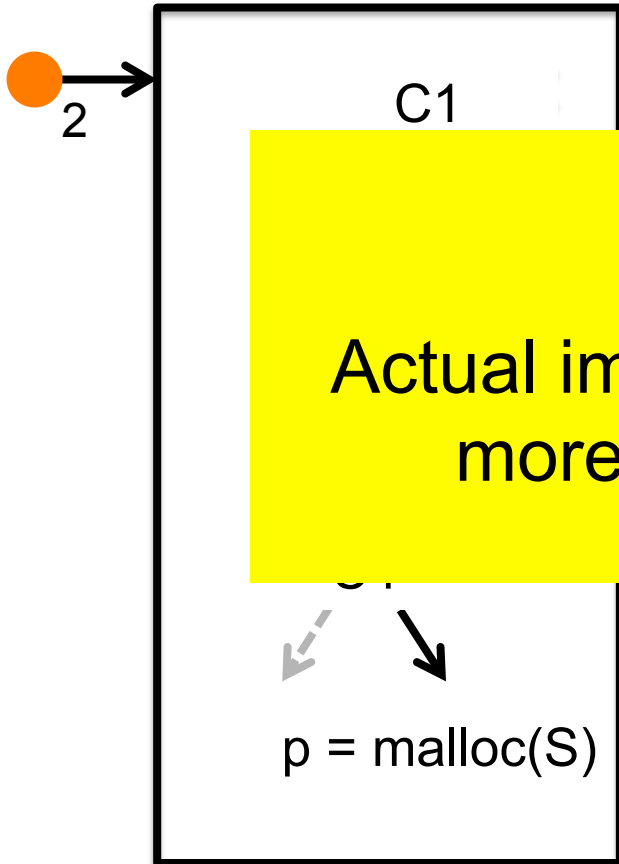
1. Satisfies C1,
2. Satisfies not C2, and
3. Overflows S

If ●<sub>2</sub> reaches malloc(S), integer overflow!

...

# What DIODE Uses (Conceptually)

display 6.5.2



## Top Down Branch Enforcement

- Solver finds new input ●<sub>1</sub> that
- 1. Satisfies C1 and

### Warning

Actual implementation is significantly more complicated than this...

- 2. Satisfies not C2, and
- 3. Overflows S

If ●<sub>2</sub> reaches malloc(S), integer overflow!

...

erflow!  
●<sub>2</sub> that

# DIODE Integer Overflow Results

Application	Format	Error Location	Defect Detection Time	Enforced Branches	Error Source
cwebp 0.3.1	jpeg	jpegdec.c:248	10 sec	0	New
dillo 2.1	png	png.c:203	11 min	3	CVE-2009-2294
dillo 2.1	png	ftkimagebuf.cc:39	15 min	3	New
display 6.5.2	tiff	xwindow.c:5619	10 sec	0	CVE-2009-1882
display 6.5.2	tiff	cache.c:3717	10 sec	0	New
swfplay 0.5.5	swf	jpeg_rgb_decoder.c: 192	10 sec	0	New
swfplay 0.5.5	swf	jpeg.c:192	48 min	5	BuzzFuzz

**7** Integer Overflow Errors

**4** Previously Unknown

**3** Require Enforced Branches

# Tractable Numbers of Candidate Checks and Insertion Points

Application	Format	Error Location	Donor	Patch Generation Time	Candidate Checks	Insertion Points
cwebp 0.3.1	jpeg	jpegdec.c:248	feh 2.9.3	2 min	8	38
			mtpaint 3.4.0	5 min	7	214
			viewnior 1.4	12 min	1	38
dillo 2.1	png	png.c:203	feh 2.9.3	8 min	5	21
			mtpaint 3.4.0	10 min	1	21
			viewnior 1.4	17 min	1	21
		ftkimagebuf.cc:39	feh 2.9.3	4 min	5	33
			mtpaint 3.4.0	10 min	1	33
			viewnior 1.4	13 min	1	33
display 6.5.2	tiff	xwindow.c:5619	feh 2.9.3	8 min	20	74
			viewnior 1.4	15 min	10	74
		cache.c:803	feh 2.9.3	4 min	20	49
			viewnior 1.4	15 min	10	49
swfplay 0.5.5	swf	jpeg_rgb_decoder.c:253	gnash	12 min	8	43
		jpeg.c:192	gnash	25 min	3	38



# Buffer Overflow Errors

# jasper Buffer Overflow Error

- jasper converts jpeg2K files
- jpeg2K images can have multiple tiles
- jasper has off by one error in tile handling code (checks for  $>$ , not  $\geq$ )

Off by one check  
(should be  $\geq$ )

```
if (JAS_CAST(int, sot->tleno) > dec->numtiles) {  
    jas_eprintf("invalid tile number in SOT marker segment\n");  
    return -1;  
}  
/* Set the current tile. */  
dec->curtile = &dec->tiles[sot->tleno];
```

Buffer overflow

# jasper Buffer Overflow Error

- jasper converts jpeg2K files
- jpeg2K images can have multiple tiles
- jasper has off by one error in tile handling code

Donor

openjpg 1.5.2 source

```
if ((tileno < 0) ||  
    (tileno >= (cp->tw*cp->th))) {  
    ...  
    return;  
}
```

Recipient

jasper 1.701.0 patch

```
if (!(!(dec->numtiles <= sot->tileno))) {  
    exit(-1);  
}
```

Correct Check



# gif2tiff Buffer Overflow Error

- gif specification defines max code size is 12
- gif2tiff does not check
- if code size > 12, buffer overflow

Donor

display 6.5.2-9 source

```
#define MaximumLZWBits 12
if (data_size > MaximumLZWBits)
    ThrowBinaryException(...);
```

Recipient

gif2tiff (libtiff 4.0.3) patch

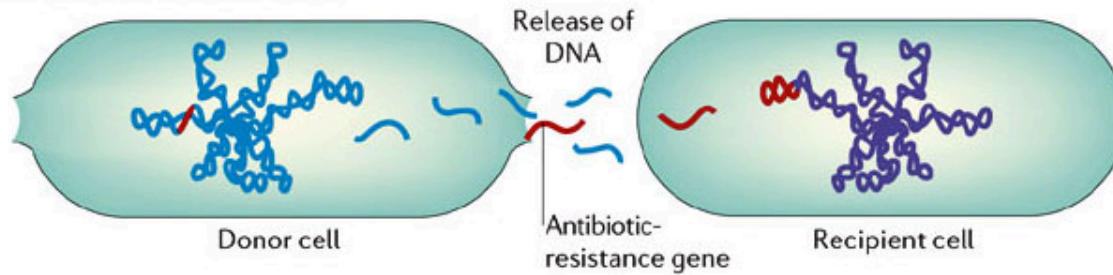
```
if (!(((unsigned char)
      (((unsigned int) datasize) <=
       ((unsigned int) 12)))))) {
    exit(-1);
}
```

# Usage Scenarios

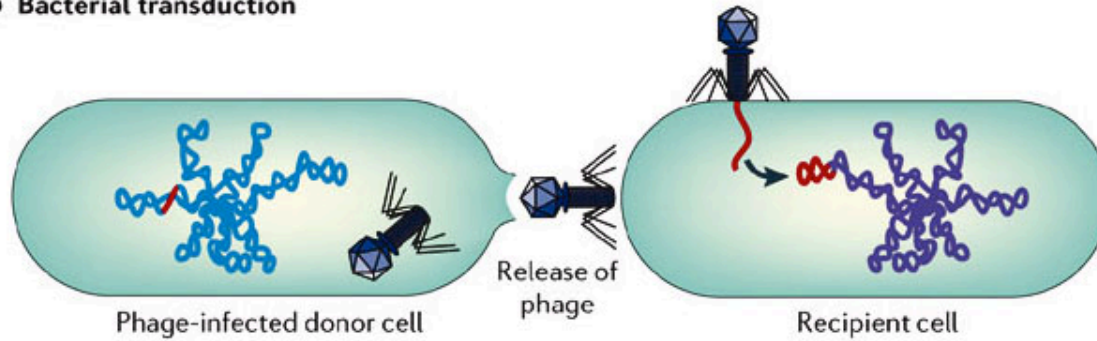
- Improving existing applications
  - Patch software defects
  - Remove security vulnerabilities
- New development approaches
  - Automatic Correctness Acquisition
    - Develop common case code (only)
    - Acquire error detection and handling code from other applications
  - Hybrid Applications  
(combine parts from multiple applications)

# Bigger Picture

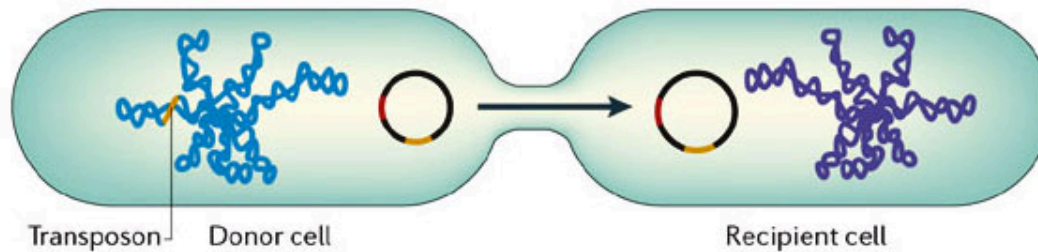
## a Bacterial transformation



## b Bacterial transduction



## c Bacterial conjugation



# CodePhage (CP)



Stelios  
Sidiroglou-Douskos



Eric Lahtinen

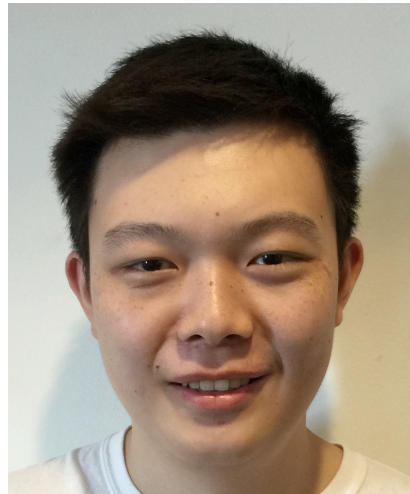


Fan Long

# RC2



Fan Long



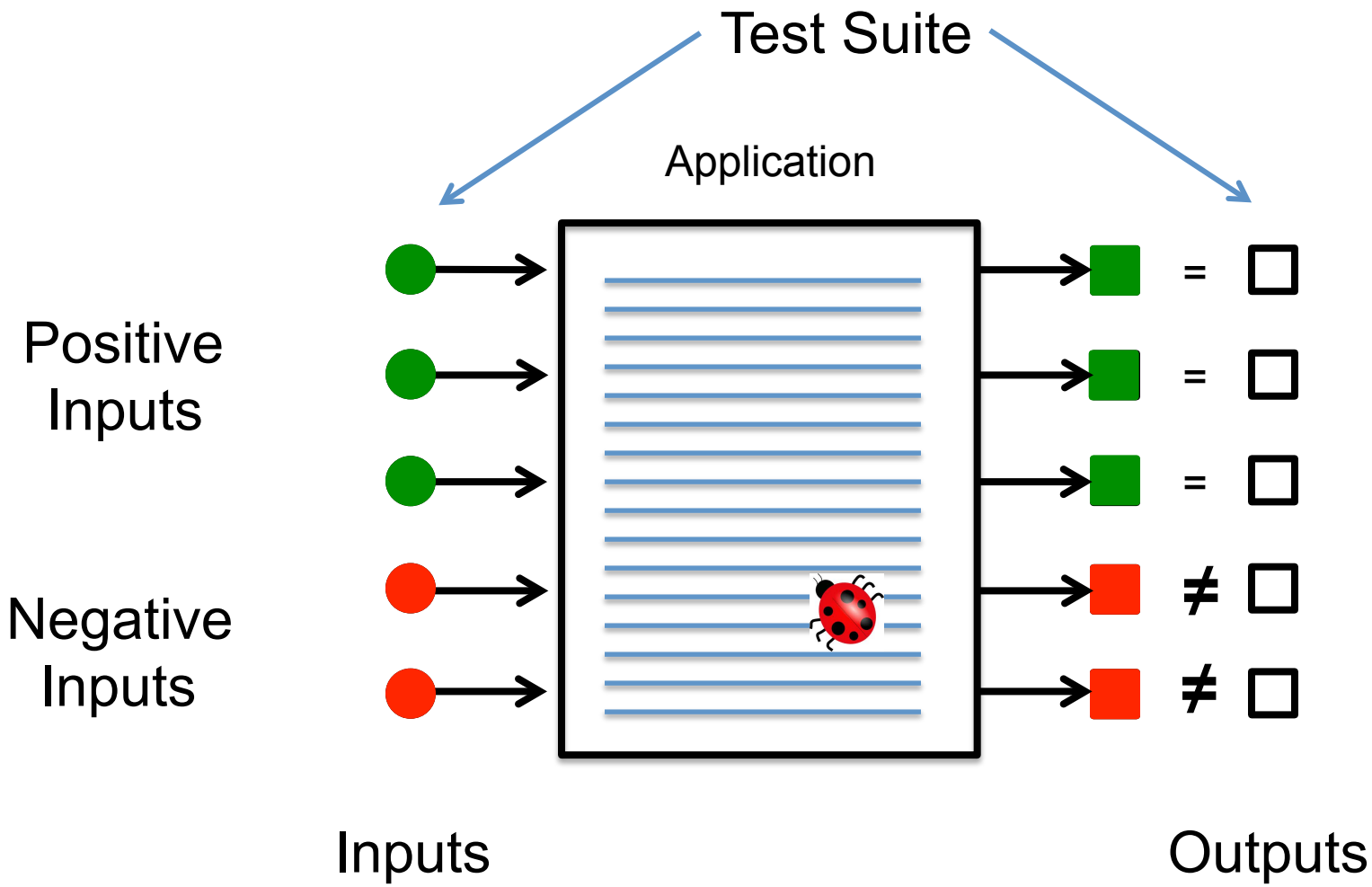
Zichao Qi



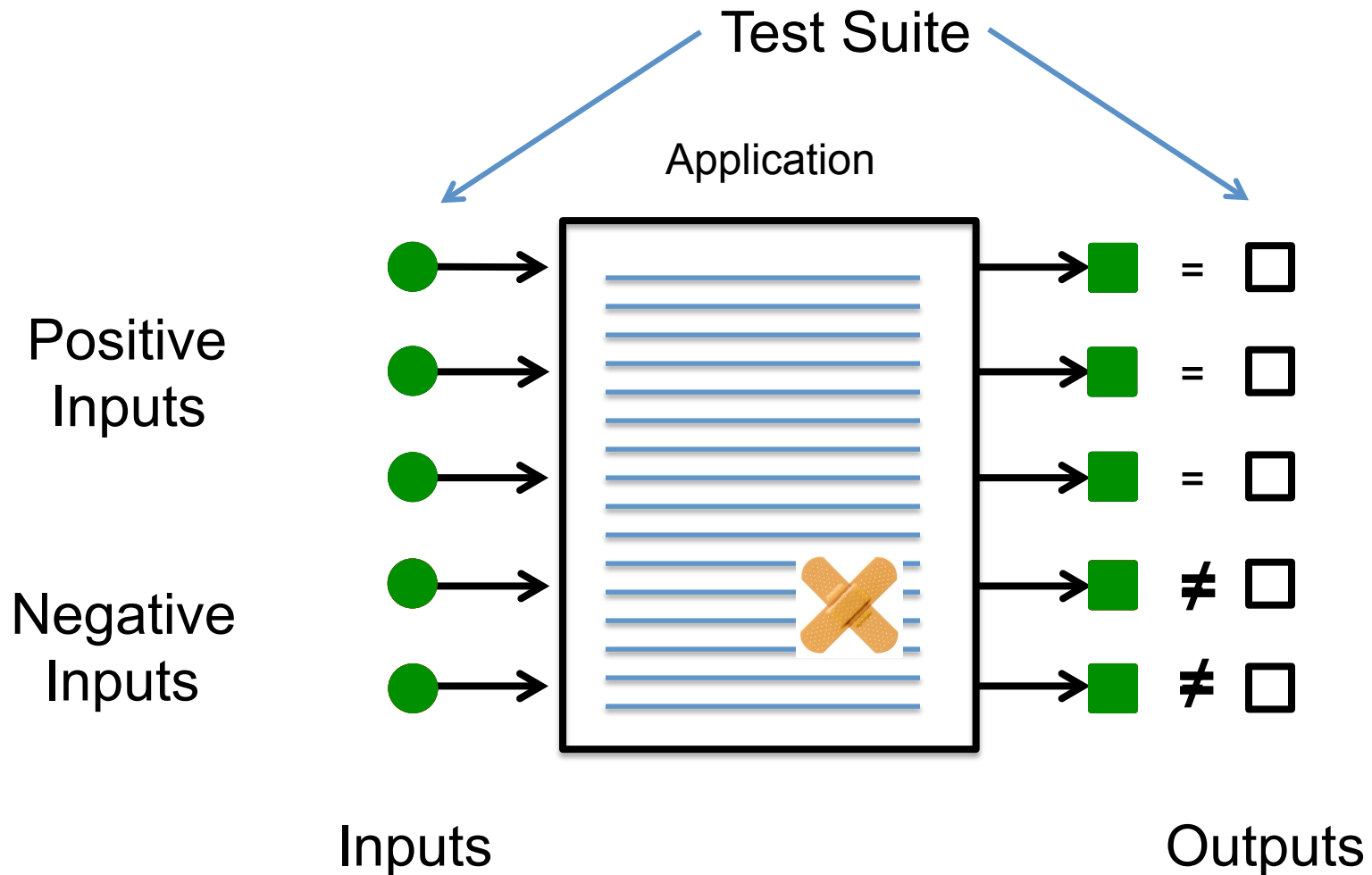
Sara Achour



# RC2 Setup



# RC2 Goal



# First Step: Defect Localization

## Statement Priority

- Statements executed by negative inputs
- Statements not executed by positive inputs
- Statements executed late in execution

$$N(s) = | \{ i \in \text{Negative. } i \text{ executes } s \} |$$

$$P(s) = | \{ i \in \text{Positive. } i \text{ executes } s \} |$$

$$L(s) = \sum_{i \in \text{Neg}} \text{index of last execution of } s$$

# Defect Localization Priority

$s1 > s2$  if

$N(s1) > N(s2)$  or

$N(s1)=N(s2)$  and  $P(s1) < P(s2)$  or

$N(s1)=N(s2)$  and  $P(s1)=P(s1)$  and  $L(s1) > L(s2)$

$N(s) = | \{ i \in \text{Negative. } i \text{ executes } s \} |$

$P(s) = | \{ i \in \text{Positive. } i \text{ executes } s \} |$

$L(s) = \sum_{i \in \text{Neg}} \text{index of last execution of } s$

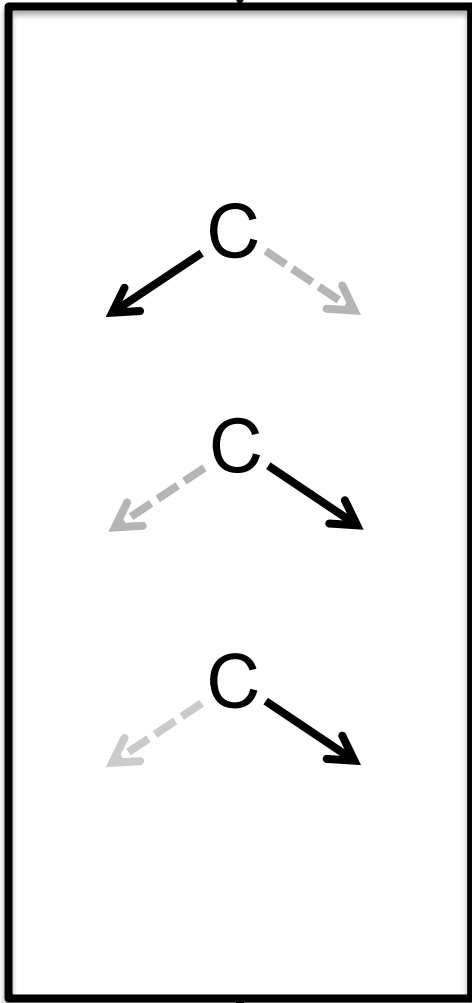
# Using Defect Localization Priority

- Priority of  
    **if (C) { S1 } else { S2 }**  
is maximum priority of **S1, S2**
- Search space is 500 highest priority statements

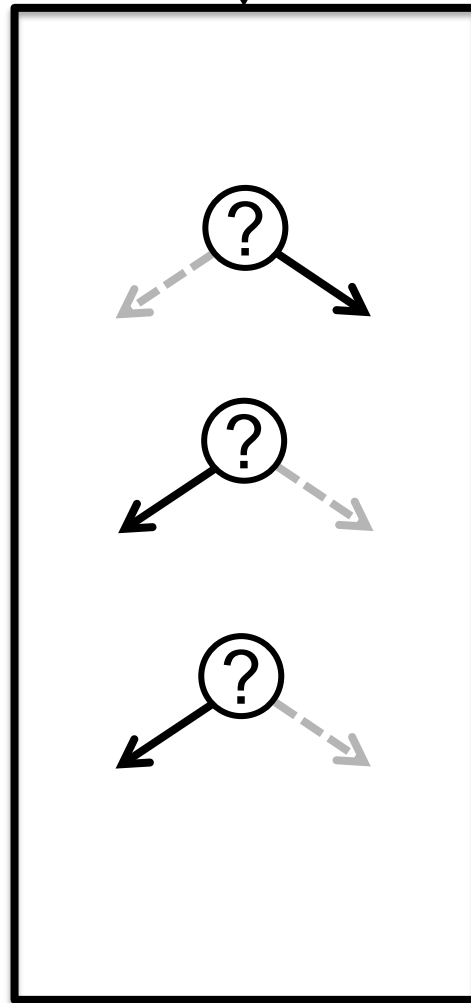
# Condition Synthesis

- Defect localizer identifies an if statement  
if (C) { ... } else { ... }
- Consider two kinds of patches  
if (C || **E**) { ... } else { ... } (loosen)  
if (C && **E**) { ... } else { ... } (tighten)
- Two steps
  - Identify branch direction sequence that produces correct output on all inputs
  - Synthesize **E** that generates (close to) that sequence

Original Application



Negative Input

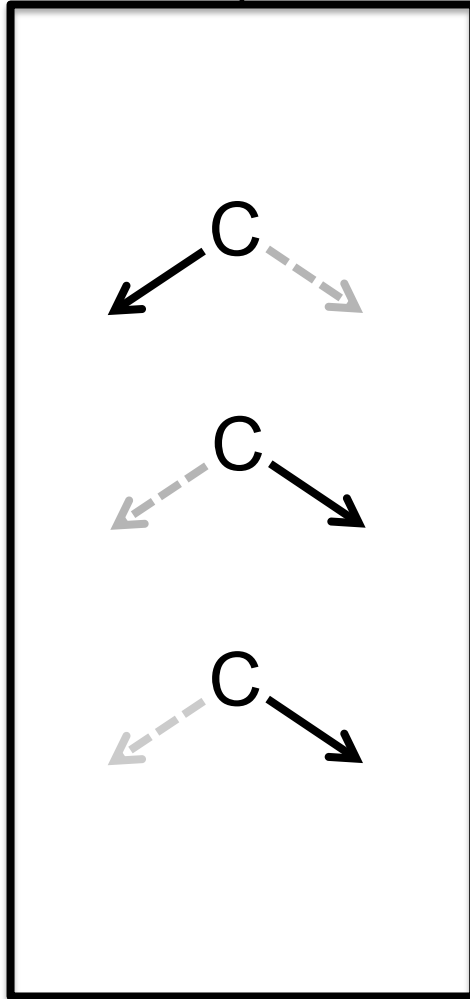


Replay Shimmed Application

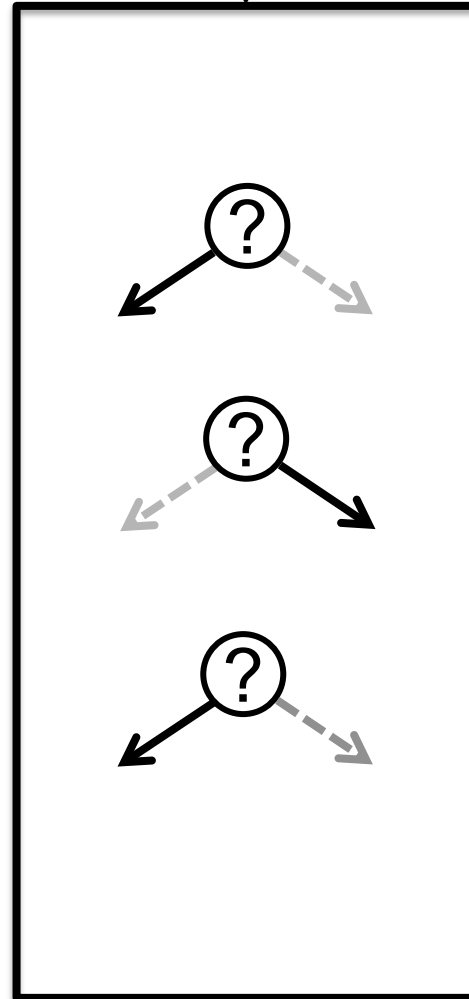
Record Branch Directions

Flip Branch Directions

Original  
Application



Negative  
Input



Replay  
Shimmed  
Application

Record  
Branch  
Directions

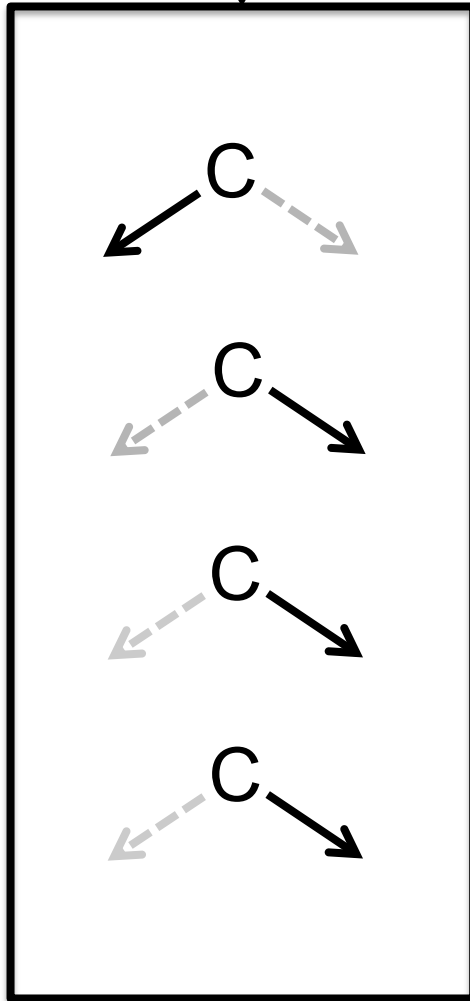
Flip  
Last  
Branch  
Direction(s)



Original Application



Record Branch Directions

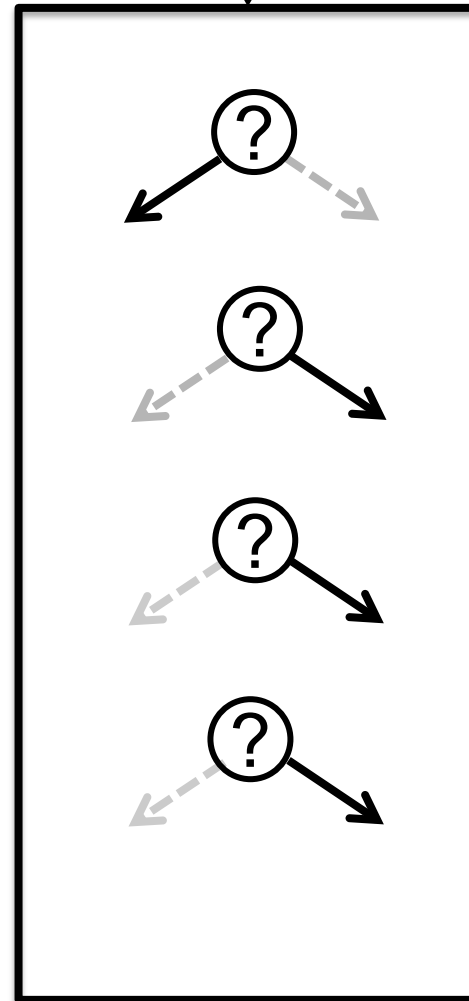


Positive Input



Replay Shimmed Application

Keep Same Branch Direction



# Condition Synthesis (C is $x > 4$ )

**E is  $y = 0$**

Relevant Variables			$x > 4$	$x > 4 \parallel \mathbf{E}$
x	y	z	Original Branch Direction	Desired Branch Direction
5	1	3	T	T
2	1	7	F	F
7	1	5	T	T
9	0	2	T	T
6	2	3	T	T
1	3	9	F	F
0	0	6	<b>F</b>	<b>T</b>

Branch Directions From Positive Input

Branch Directions From Negative Input

Goal: Synthesize an **E** that provides desired branch directions

# Building On Condition Synthesis


- Defect localizer identifies **S**

- Guard insertion mutation:

**S**  **if (E) { S }**

Use condition synthesis to generate **E**

- Conditional control-flow insertion mutation:

**S**  **if (E) return C; S**

**S**  **if (E) goto label; S**

**S**  **if (E) break; S**

Use condition synthesis to generate **E**

# Another Compound Mutation

- Defect localizer identifies  $S$
- Replace subexpression mutation:
  - Variable replacement:  $S \longrightarrow S[v1/v2]$
  - Constant replacement:  $S \longrightarrow S[c1/c2]$
  - Function replacement:  
 $S \longrightarrow S[f(e1, \dots, en)/g(e1, \dots, en)]$

# Yet Another Compound Mutation

- Defect localizer identifies **S**
- Add statement via copy and replace
  - Choose some statement **Q** in program
  - Copy and Replace **Q** before **S**:

Variable replacement: **S**  **Q[v1/v2]; S**

Constant replacement: **S**  **Q[c1/c2]; S**

Function replacement:

**S**  **Q[f(e1, ..., en)/g(e1, ..., en)]; S**

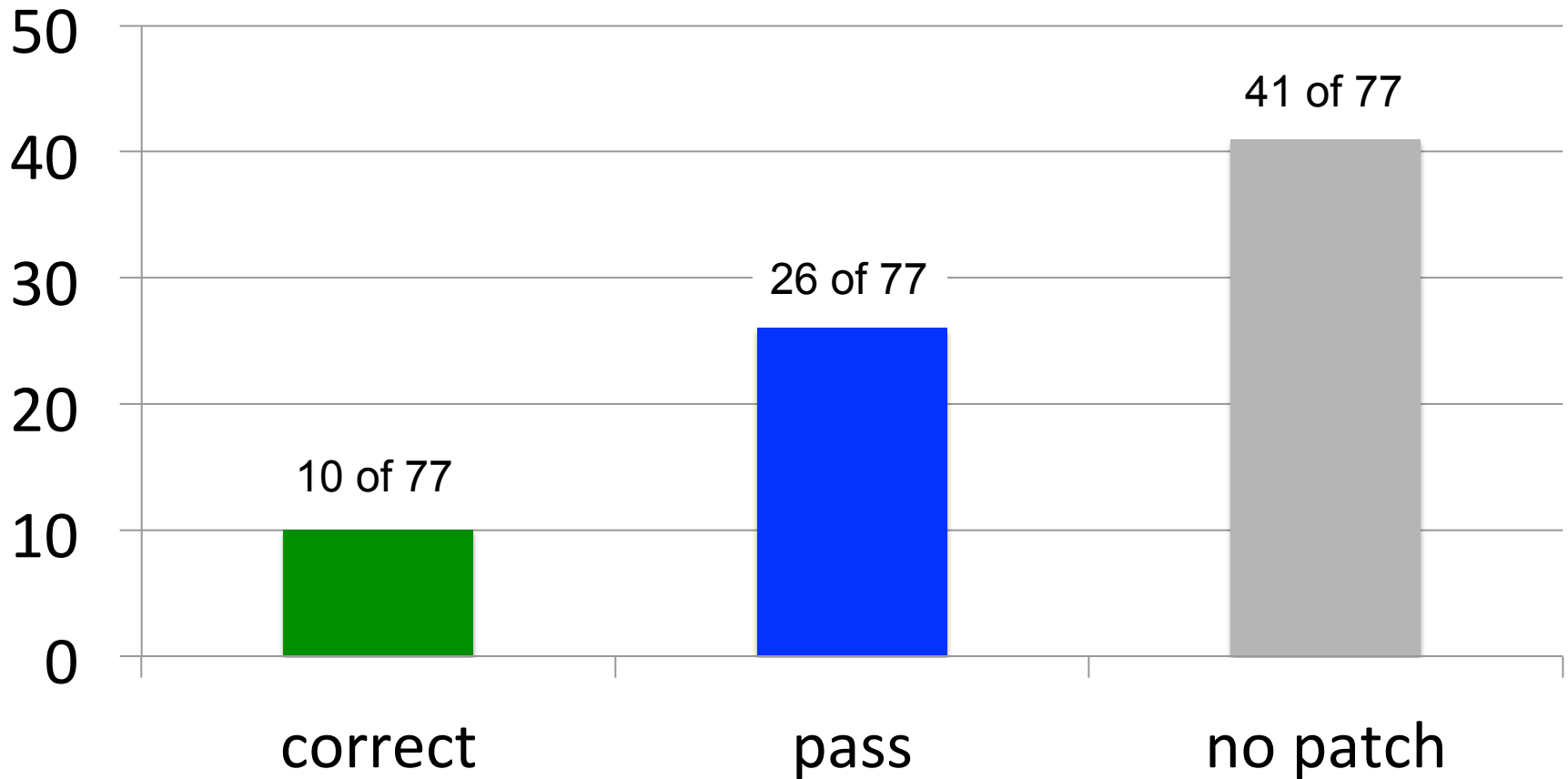
# Search Space Possibilities

- One compound mutation only
  - Simple space, relatively efficient to search
  - Targets small patches
- Multiple combined compound mutations
  - Richer space of candidate patches
  - Less efficient to search
- Current system: one compound mutation only

# Results

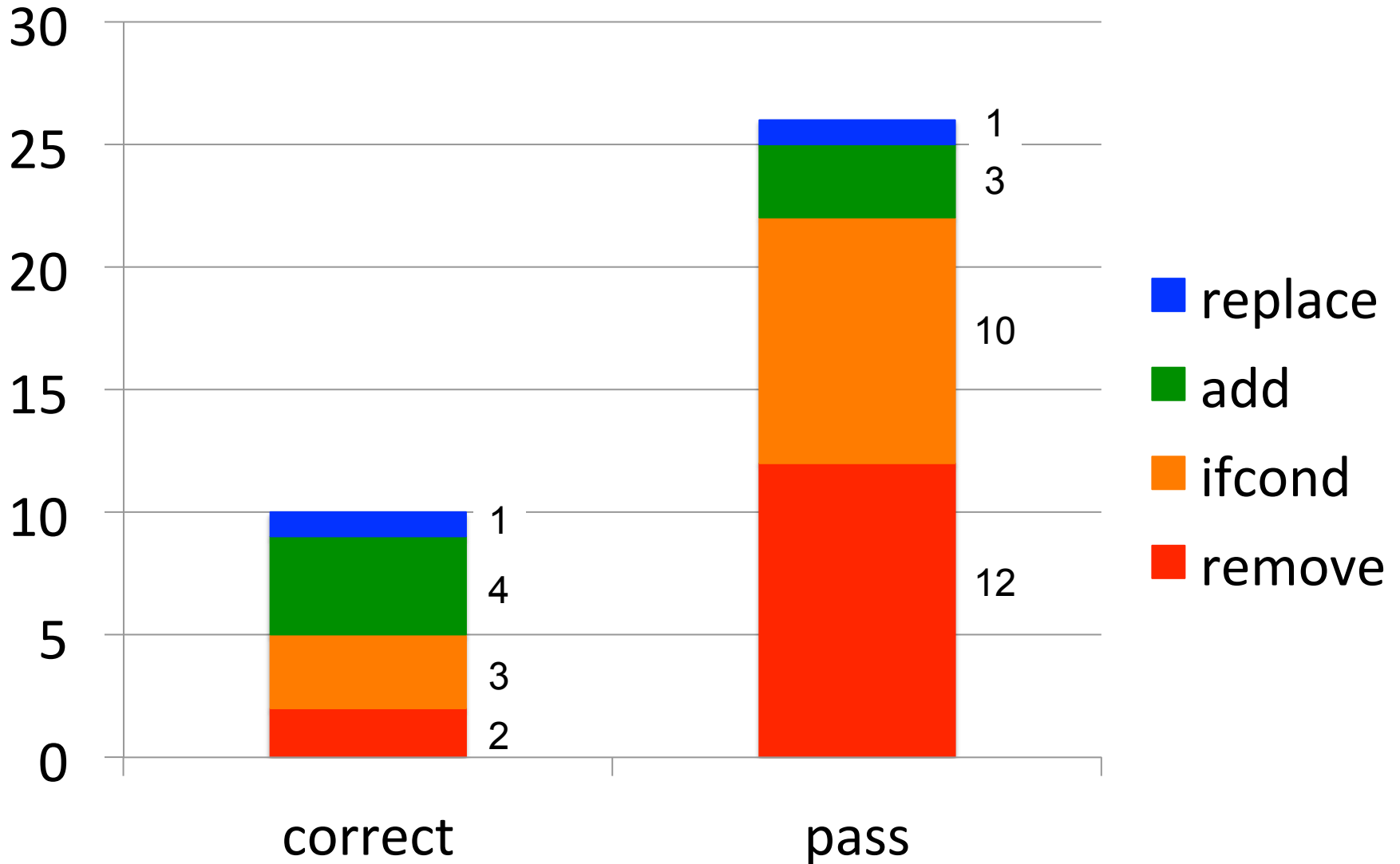
- Implemented RC2
- Evaluated on 77 defects from 3 applications (from GenProg ICSE2012 benchmarks)
  - libtiff (24 defects)
  - lighttpd (9 defects)
  - php (44 defects)
- Each defect comes with
  - Test suite that exposes defect
  - Version with developer fix applied (sometimes this is an earlier version)

# Defect Outcomes





# Patch Breakdown



# A Cautionary Example

- Defective integer overflow check:  

```
/* Check for overflow. */  
if (!dir->tdir_count || !w || (tsize_t)dir->tdir_count / w != cc)  
    goto bad;
```

  - One test case checks for incorrect input rejection
  - No test case checks for correct input rejection
- Patch simply removes check
- Reintroduces security vulnerability (CVE-2006-2025)
- Pattern: removal of unprotected functionality

# Why Not More Correct Patches?

## Two Possibilities:

1. Search space does not contain correct patch
2. Weak test suite
  - Search space does contain correct patch
  - But an earlier incorrect patch passes test suite

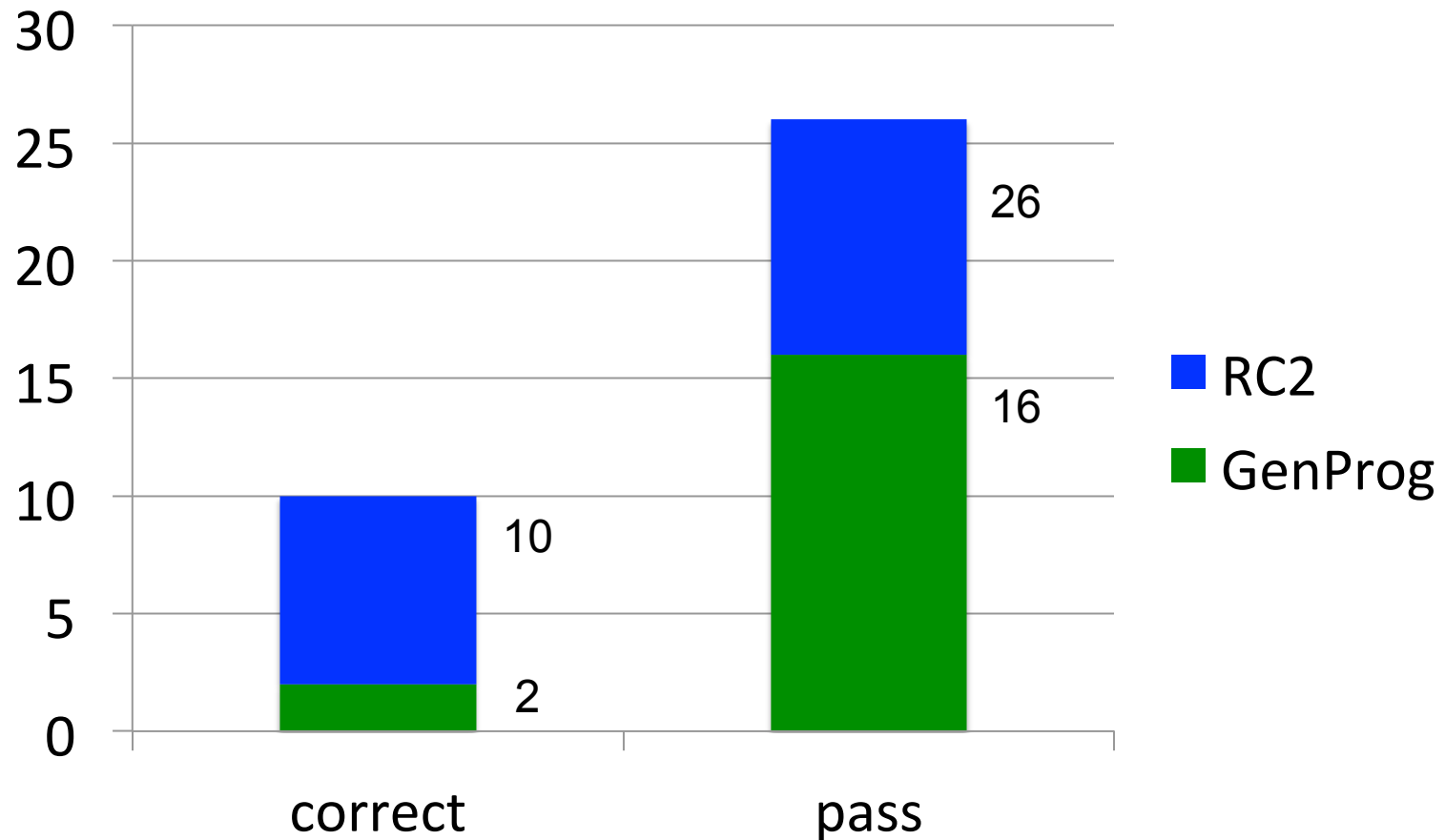
# How Many More Correct Patches Exist In Search Space?

5

Defect	Search Space	Correct Patch At	Max Test Cases	Wrong Location	Wrong Synthesis
php-A	10347	355	3	2	1
php-B	3431	22	5	0	5
php-C	5028	1996	-	-	-
libtiff-A	64821	131	13	0	13
libtiff-B	71359	7354	-	-	-

# Comparison with GenProg

A Systematic Study of Automated Program Repair: Fixing 55 out of 105 bugs for \$8 Each; Le Goues, Dewey-Vog, Forrest, Weimer; ICSE 2012



# Reasons for RC2 Success

- Separate condition synthesis from main search algorithm
  - Significantly reduces patch search space
  - Focuses search on productive part of space
- Compound mutations with replacement
  - Richer, more structured search space
  - Focuses search on productive part of space

# Two Kinds of Systems

- General
  - Basic mutations (copy, replace, remove)
  - Generate search space from multiple mutations
  - Try to find mutation that passes test suite
  - GenProg (genetic programming) [ICSE 2009, 2012]
  - RSRepair (random testing) [ICSE 2014]
- Targeted
  - Identify class of errors, build technique to eliminate error
  - FOC (out of bounds accesses) [OSDI 2004]
  - ClearView (invariant violations) [SOSP 2009]
  - Bolt (infinite loops) [OOPSLA 2012]
  - RCV (null dereference, divide by zero) [PLDI 2014]
  - PAR (human identified error patterns) [ICSE 2013]

# Pros and Cons

## General

- Pro
  - Expressive search space
  - Many potential patches
- Con
  - Search space is very large
  - Sparse patch occurrence
  - Difficult to find patches
  - Functionality removal if not protected by test case

## Targeted

- Pro
  - No (or limited) search
  - More predictable consequences
  - Hot recovery
  - No source code
- Con
  - Less general



# RC2

- More structured than general systems
  - Condition synthesis
  - Compound mutations
  - More effective search space
- Desired functionality must be protected by positive test cases
- Targeted (right now) to small patches

# Looking To The Future

- We are now starting to automate traditional software engineering tasks
  - Finding and eliminating defects
  - Transferring code between applications
- Software development today
  - Manual and slow
  - Expensive
- Starting to see how automation may qualitatively transform software our society can produce
  - Reliability and security
  - Functionality