Program Analysis for Security

John Mitchell

Software bugs are serious problems

07-31-2010, 12:57 PM

911crashes
Junior Member

Join Date: Jul 2010
Posts: 2

[OFFLINE]

calling 911 crashes my HTC evo 4G, every time

I happen to need to call 911 one night, found that my phone crashes every time I dial 911.

my wife's phone does not do that, any thought?

by the way, it is hard to test this problem due to the sensitivity of calling 911 repeatedly.
thanks,

heartboken

Thanks: Isil and Thomas Dillig

App stores

Apps for whatever you're up for.

Stay on top of the news. Stay on top of your finances. Or plan your dream vacation. No matter what you want to do with your iPhone, there's probably an app to help you do it.

Business

iPhone is ready for work. Manage projects, track stocks, monitor finances, and more with these 9-to-5 apps.

View business apps in the App Store ›

Education

Keep up with your studies using intelligent education apps like King of Math and NatureTap.

View education apps in the App Store ›

Entertainment

Kick back and enjoy the show. Or find countless other ways to entertain yourself. These apps offer hours of viewing pleasure.

View entertainment apps in the App Store ›

Family & Kids

Turn every night into family night with interactive apps that are fun for the whole house.

View family and kids apps in the App Store ›

Finance

Create budgets, pay bills, and more with financial apps that take everything into account.

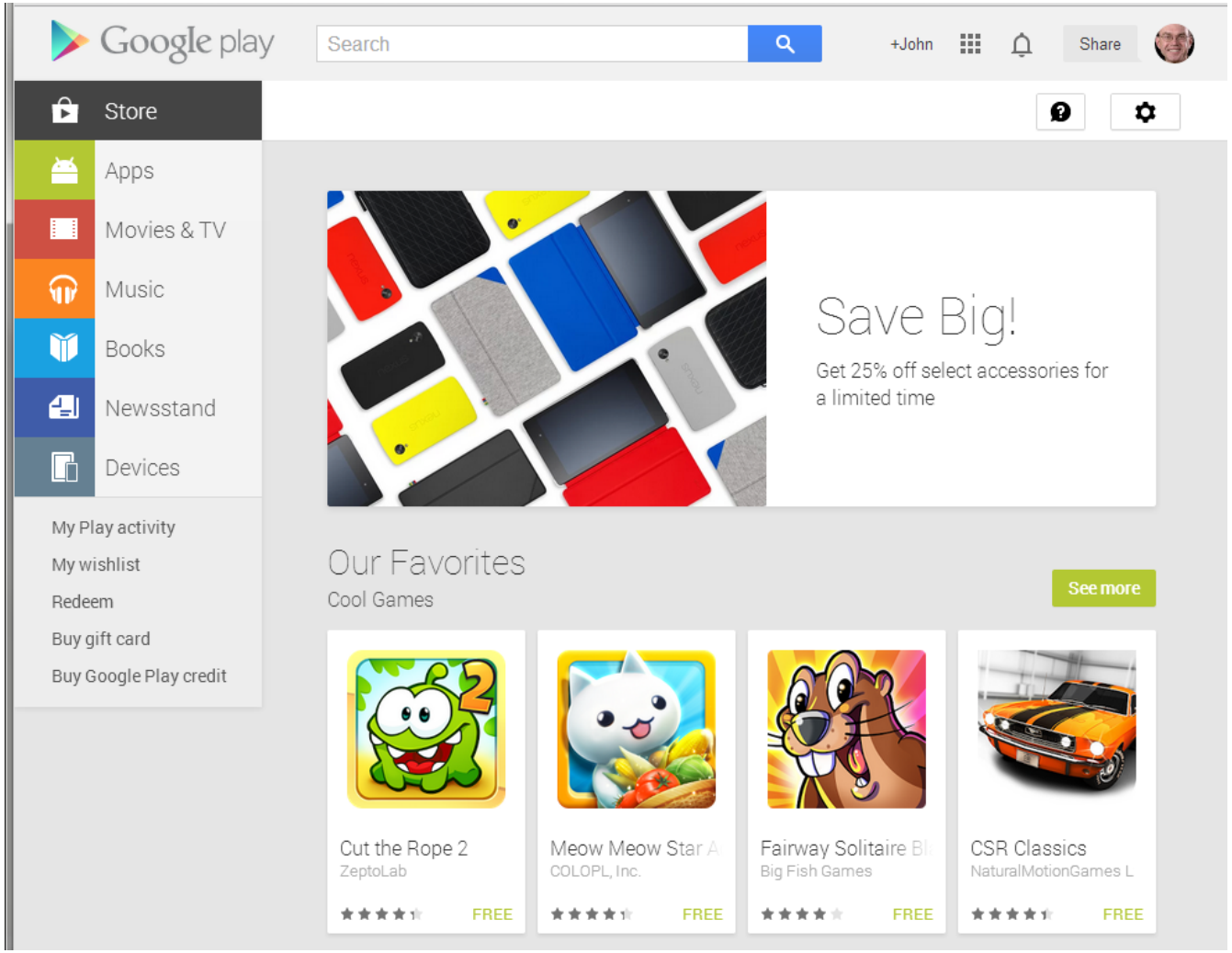
View finance apps in the App Store ›

Food & Drink

Hungry? Thirsty? A little of both? Learn new recipes, drinks, and the secrets behind what makes a great meal.

View food and drink apps in the App Store ›

App stores

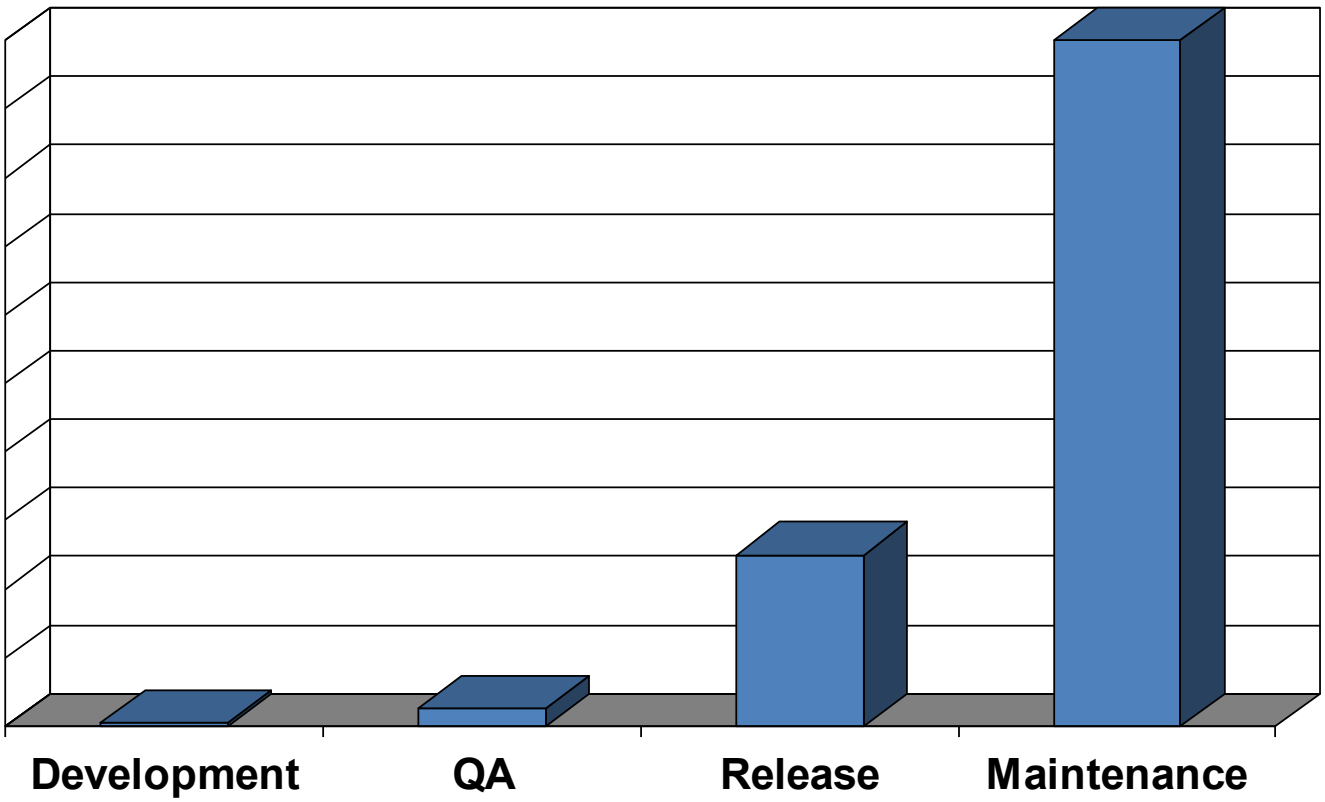

Google play
Search
+John
Share
Store
Apps
Movies & TV
Music
Books
Newsstand
Devices
My Play activity
My wishlist
Redeem
Buy gift card
Buy Google Play credit
Save Big!
Get 25% off select accessories for a limited time
Our Favorites
Cool Games
See more
Cut the Rope 2
ZeptoLab
FREE
Meow Meow Star A
COLOPL, Inc.
FREE
Fairway Solitaire Bl
Big Fish Games
FREE
CSR Classics
NaturalMotionGames L
FREE

How can you tell whether software you

- Buy
- Develop
- Install

is safe to run?

Cost of Fixing a Defect



Credit: Andy Chou, Coverity

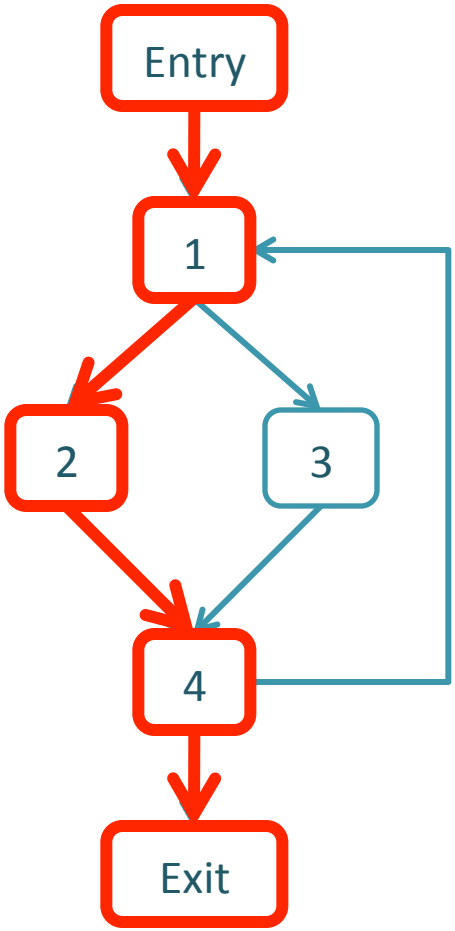
Several approaches

- Instrument code for testing
  - Heap memory: Purify
  - Perl tainting (information flow)
  - Java race condition checking
- Black-box testing
  - Fuzzing and penetration testing
  - Black-box web application security analysis
- Static code analysis
  - FindBugs, Fortify, Coverity, MS tools, …

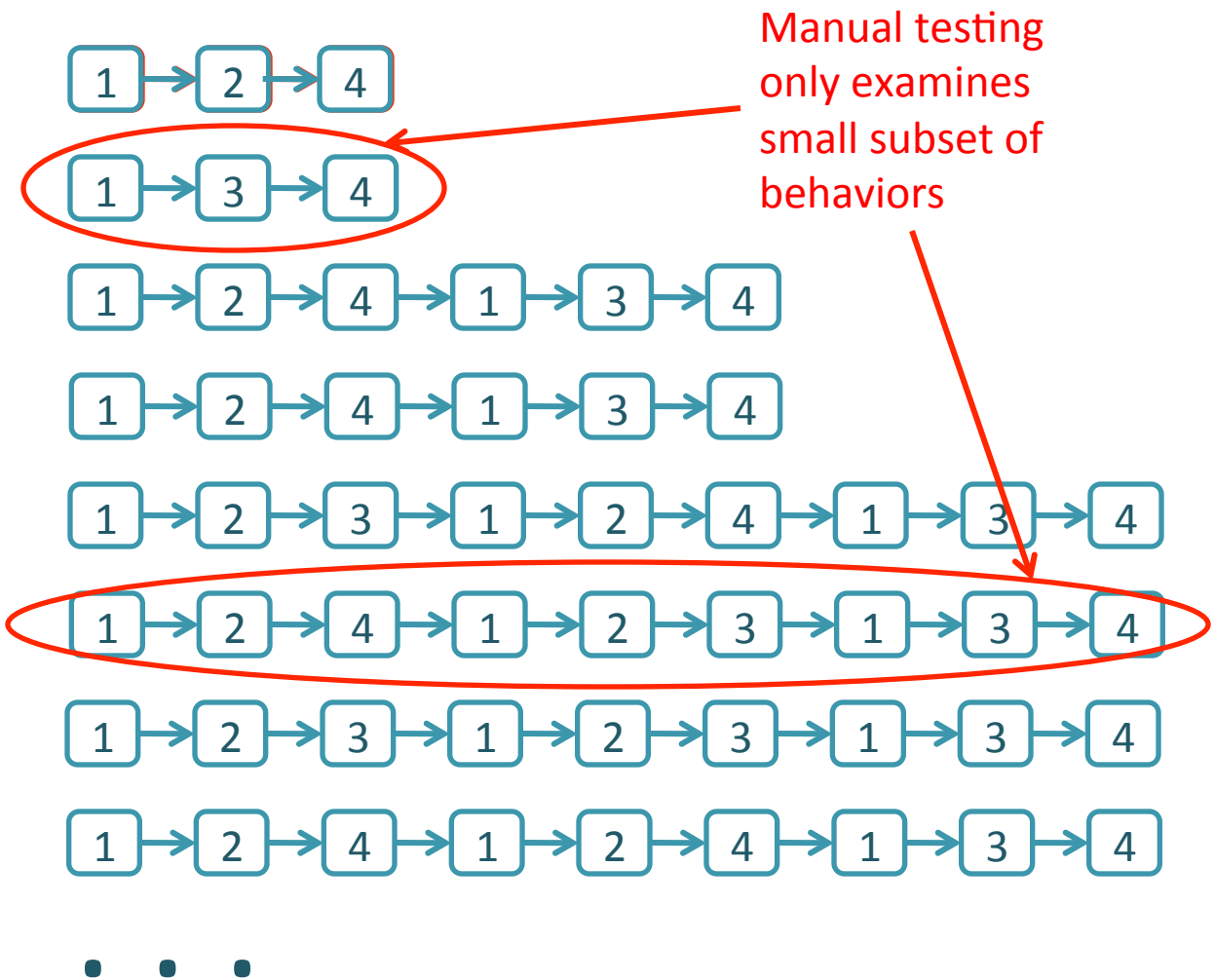
Outline

- General discussion of static analysis tools
  - Goals and limitations
  - Approach based on abstract states
- More about one specific approach
  - Property checkers from Engler et al., Coverity
  - Sample security-related results
- Static analysis for of Android apps

Slides from: S. Bugrahe, A. Chou, I&T Dillig, D. Engler, J. Franklin, A. Aiken, …

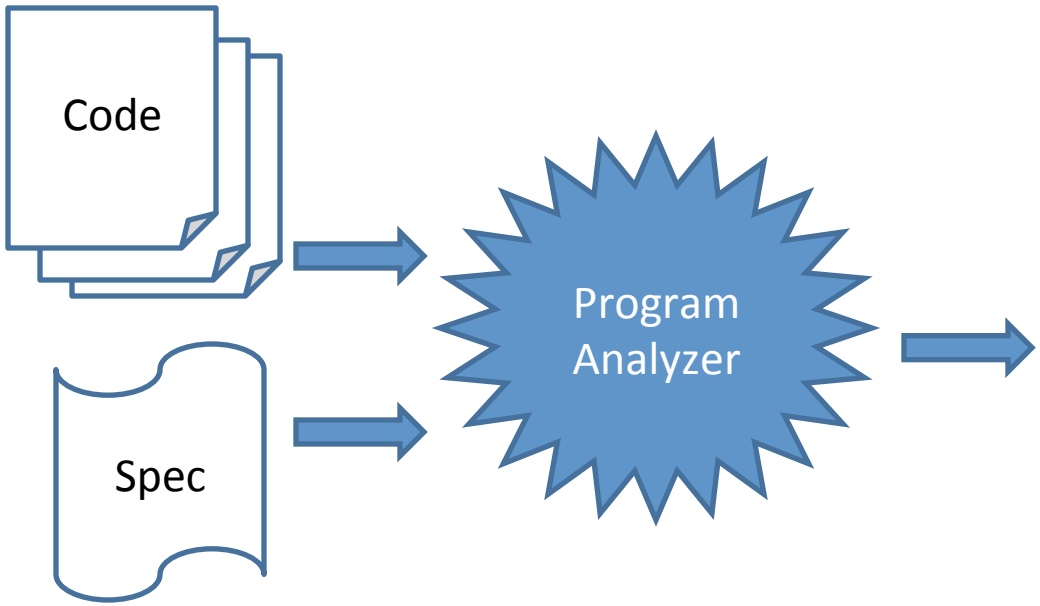


Software



Behaviors

Program Analyzers



Report	Type	Line
1	mem leak	324
2	buffer oflow	4,353,245
3	sql injection	23,212
4	stack oflow	86,923
5	dang ptr	8,491
…	…	…
10,502	info leak	10,921

false alarm

false alarm

analyze large code bases

potentially reports many warnings

may emit false alarms

Example

- Coverity demo video: http://youtu.be/_Vt4niZfNeA
- Observations
  - Code analysis integrated into development workflow
  - Program context important: analysis involves sequence of function calls, surrounding statements
  - This is a sales video: no discussion of false alarms

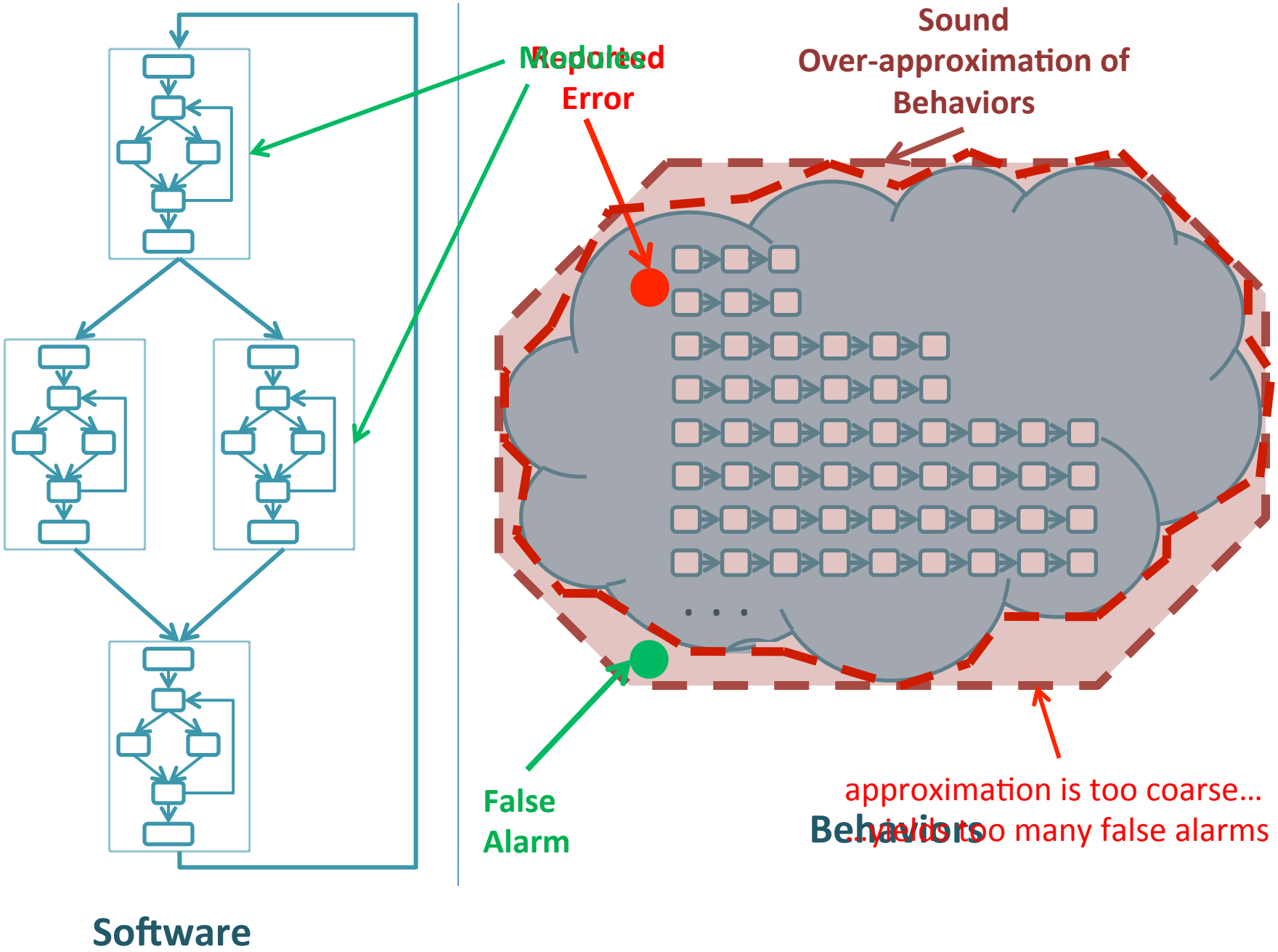
Static analysis goals

- Bug finding
  - Identify code that the programmer wishes to modify or improve
- Correctness
  - Verify the absence of certain classes of errors

Soundness, Completeness

Property	Definition
Soundness	If the program contains an error, the analysis will report a warning. "Sound for reporting correctness"
Completeness	If the analysis reports an error, the program will contain an error. "Complete for reporting correctness"

	Complete	Incomplete
Sound	Reports all errors Reports no false alarms **Undecidable**	Reports all errors May report false alarms **Decidable**
Unsound	May not report all errors Reports no false alarms **Decidable**	May not report all errors May report false alarms **Decidable**

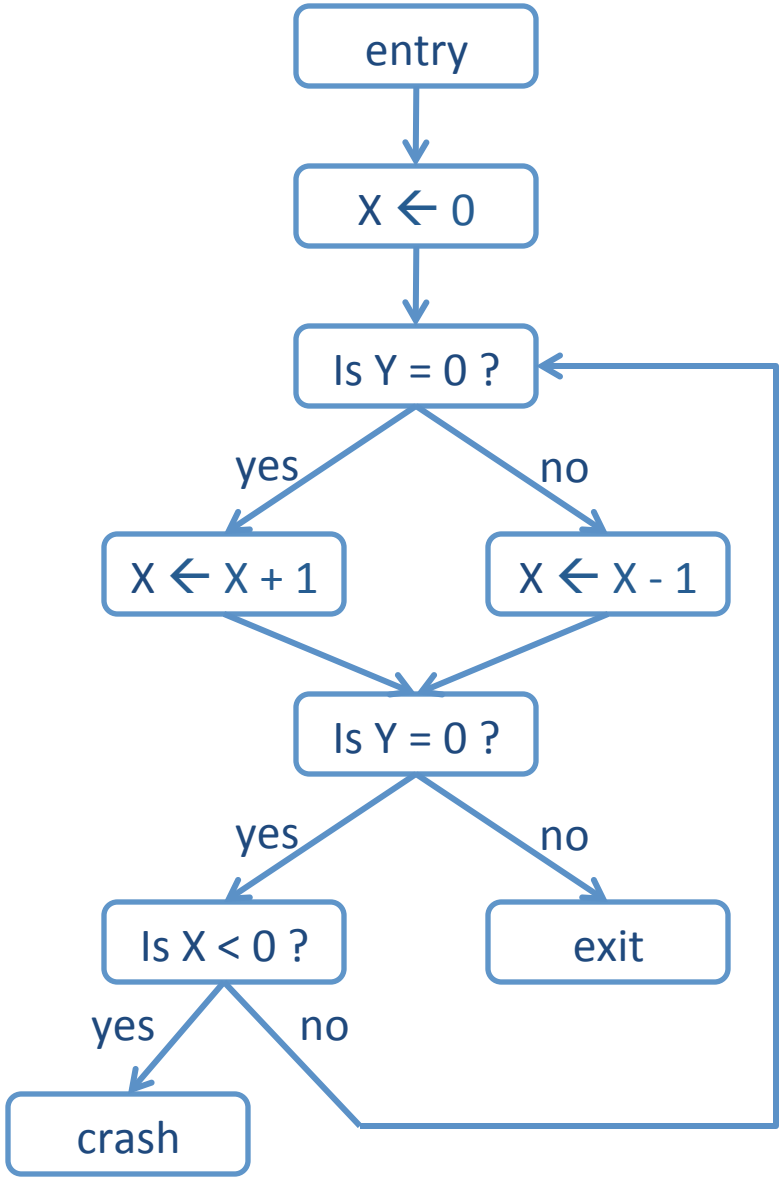

Reported
Error
Sound
Over-approximation of
Behaviors
False
Alarm
approximation is too coarse...
yields too many false alarms
Behaviors
Software

Outline

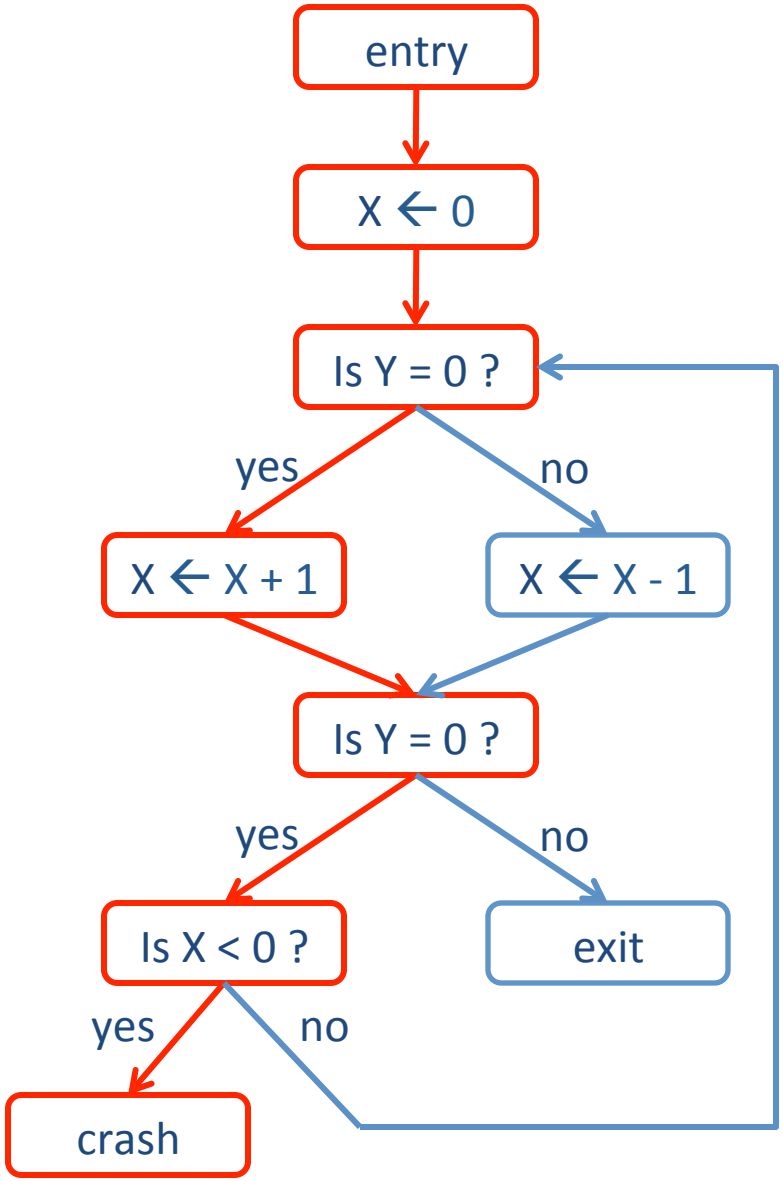
- General discussion of tools
  - Goals and limitations
  - ➡ Approach based on abstract states
- More about one specific approach
  - Property checkers from Engler et al., Coverity
  - Sample security-related results
- Static analysis for Android malware
  - …

Slides from: S. Bugrahe, A. Chou, I&T Dillig, D. Engler, J. Franklin, A. Aiken, …

Does this program ever crash?



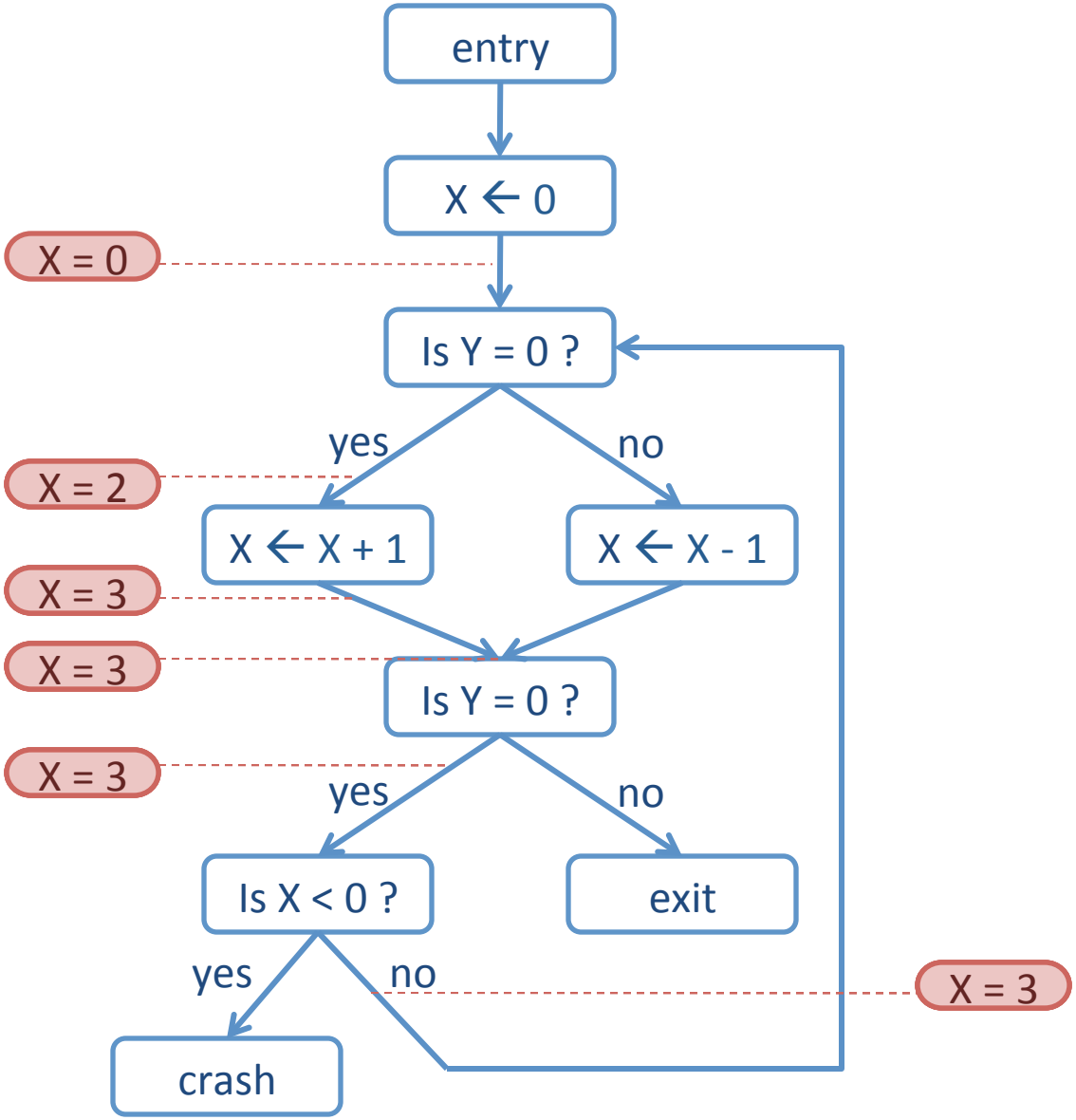
Does this program ever crash?



infeasible path!

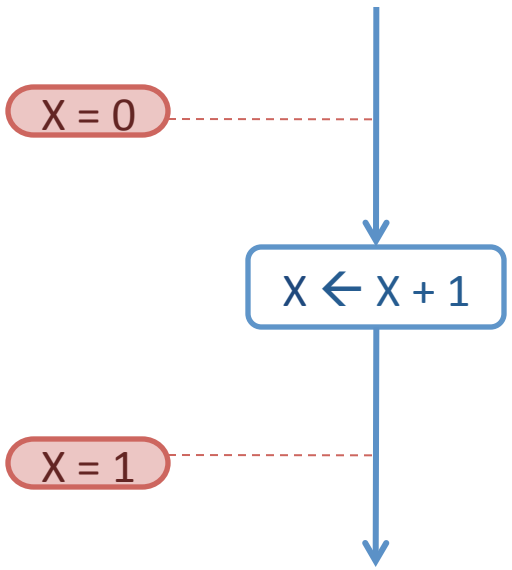
… program will never crash

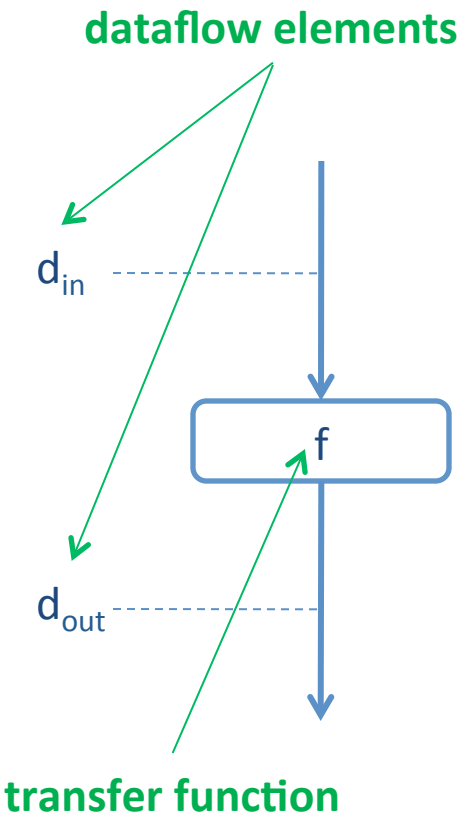
Try analyzing without approximating…



non-termination!

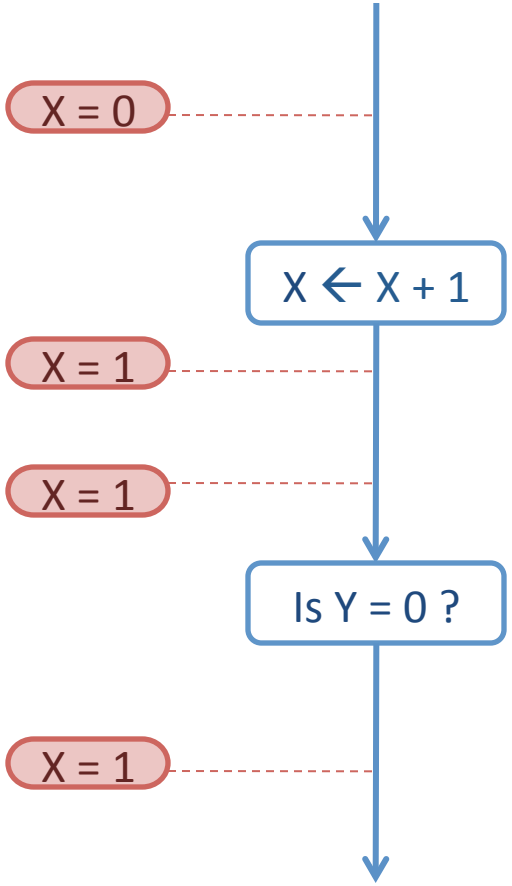
… therefore, need to approximate

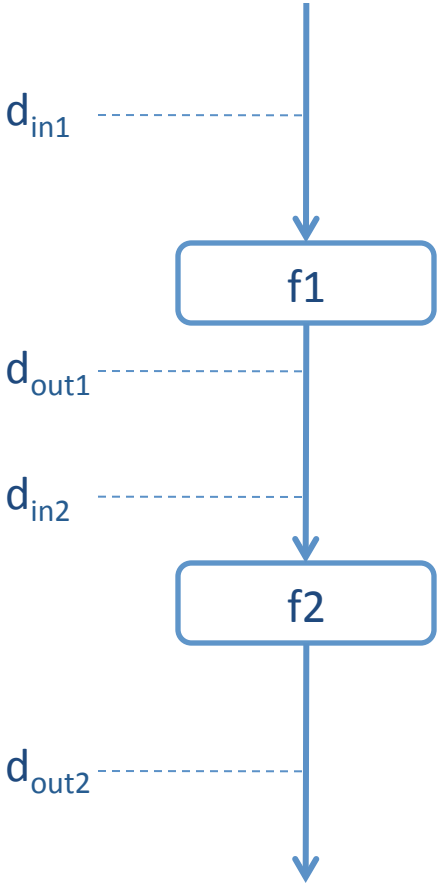




$$d_{out} = f(d_{in})$$

dataflow equation

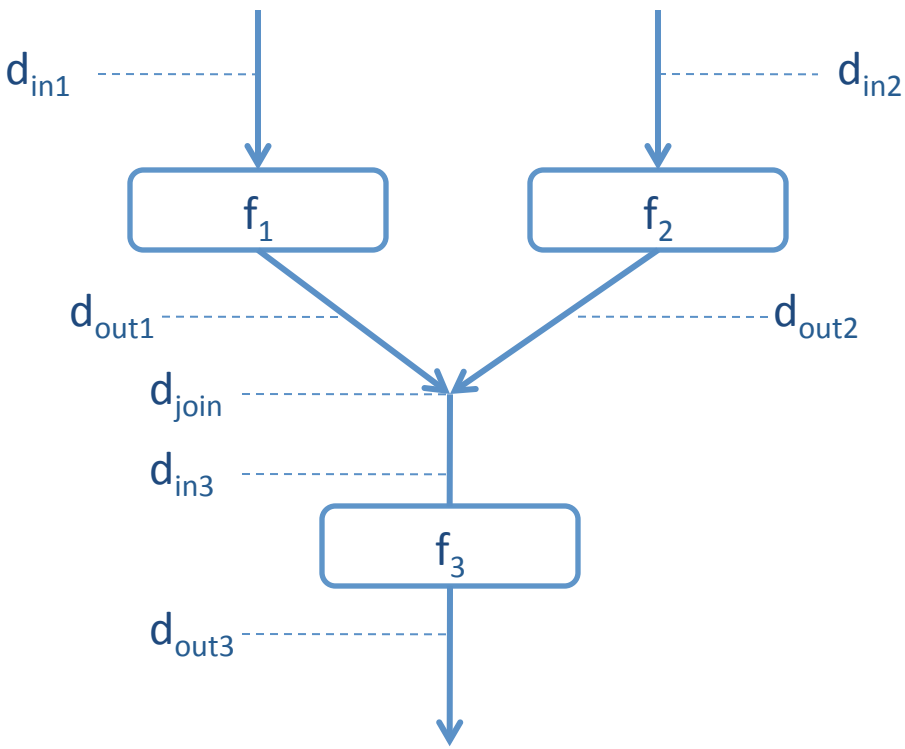




$$d_{out1} = f_1(d_{in1})$$

$$d_{out1} = d_{in2}$$

$$d_{out2} = f_2(d_{in2})$$



$$d_{out1} = f_1(d_{in1})$$

$$d_{out2} = f_2(d_{in2})$$

$$d_{join} = d_{out1} \sqcup d_{out2}$$

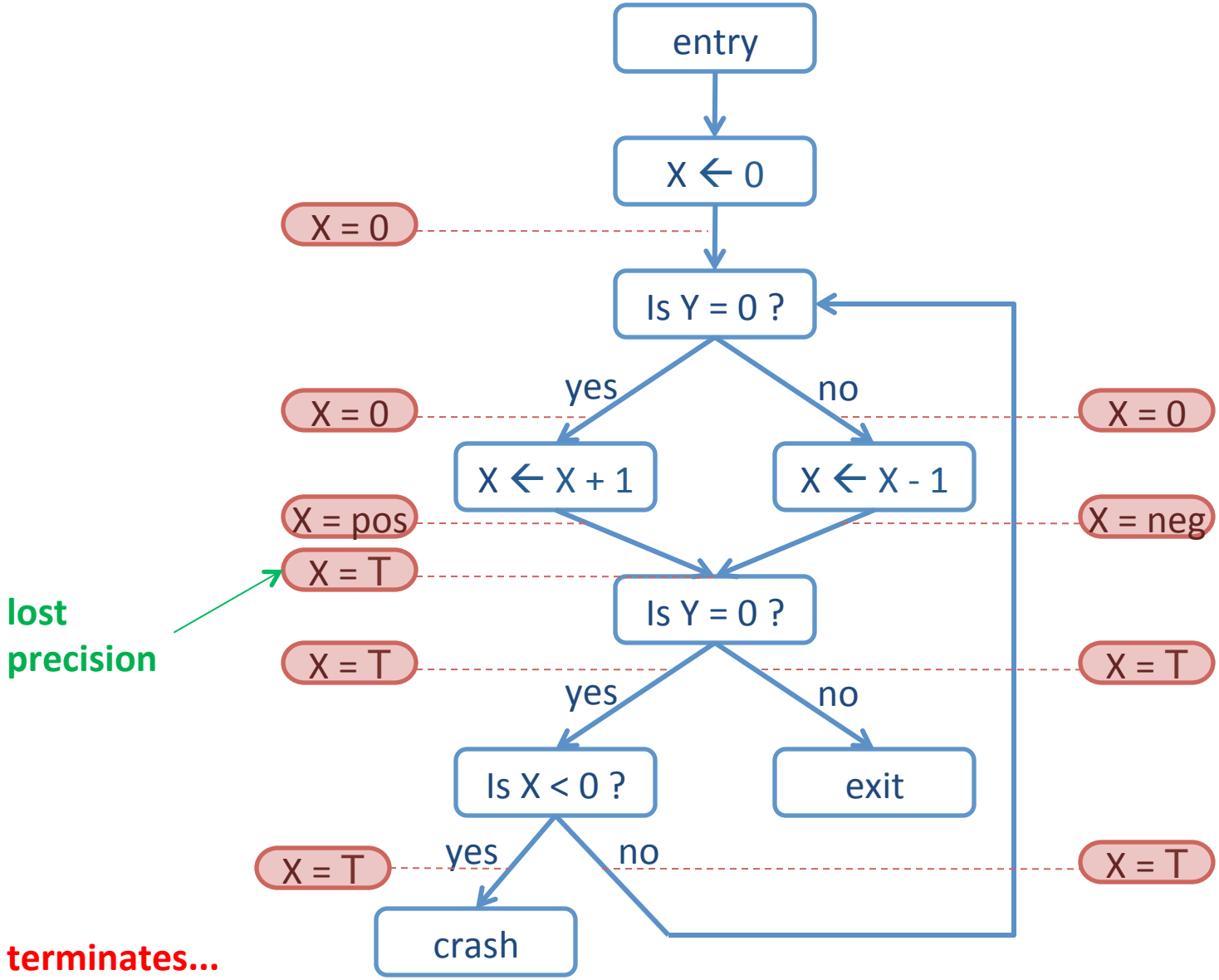
$$d_{join} = d_{in3}$$

$$d_{out3} = f_3(d_{in3})$$

least upper bound operator
Example: union of possible values

What is the space of dataflow elements, Δ?
What is the least upper bound operator, $\sqcup$?

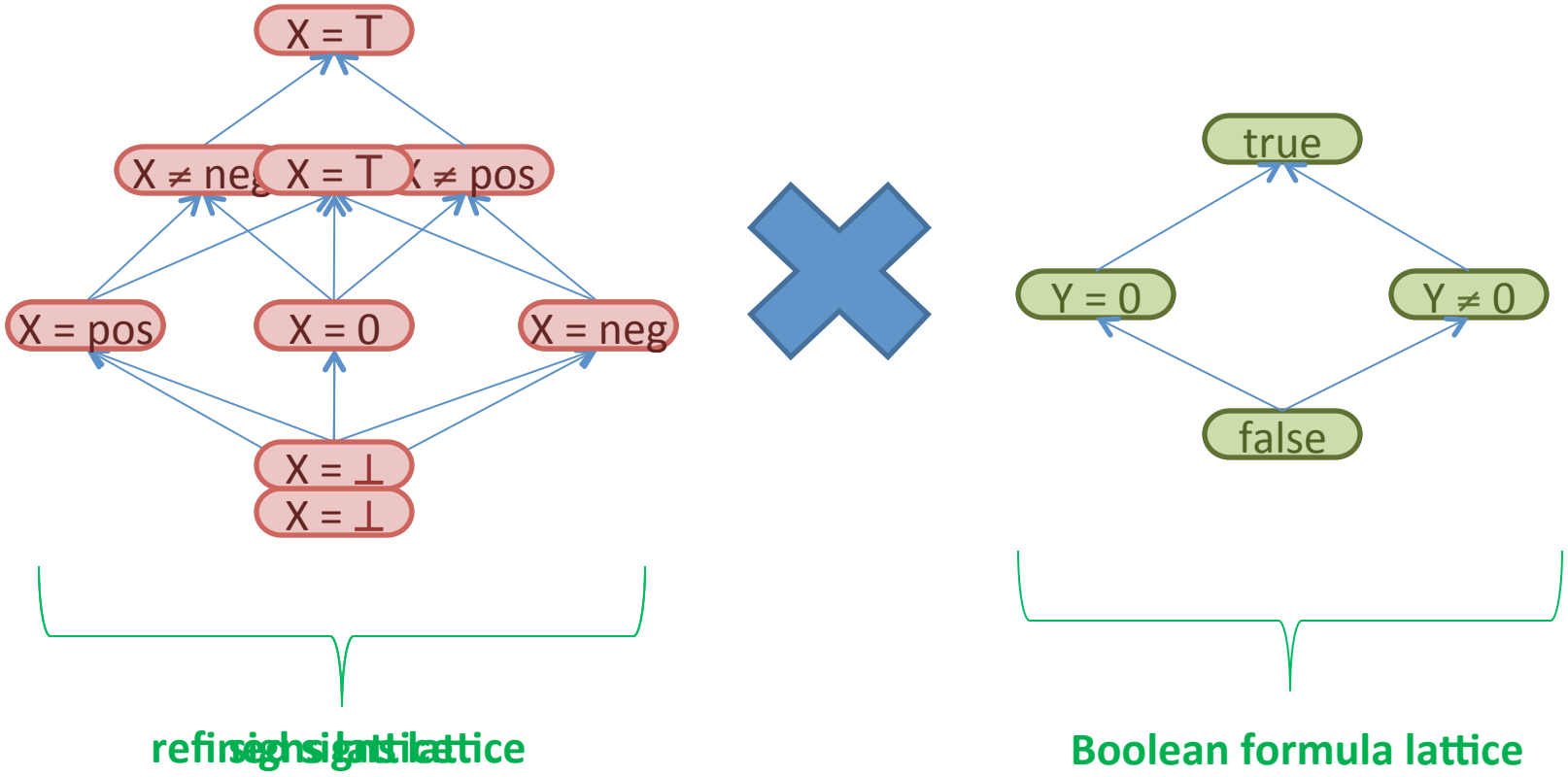
Try analyzing with “signs” approximation…



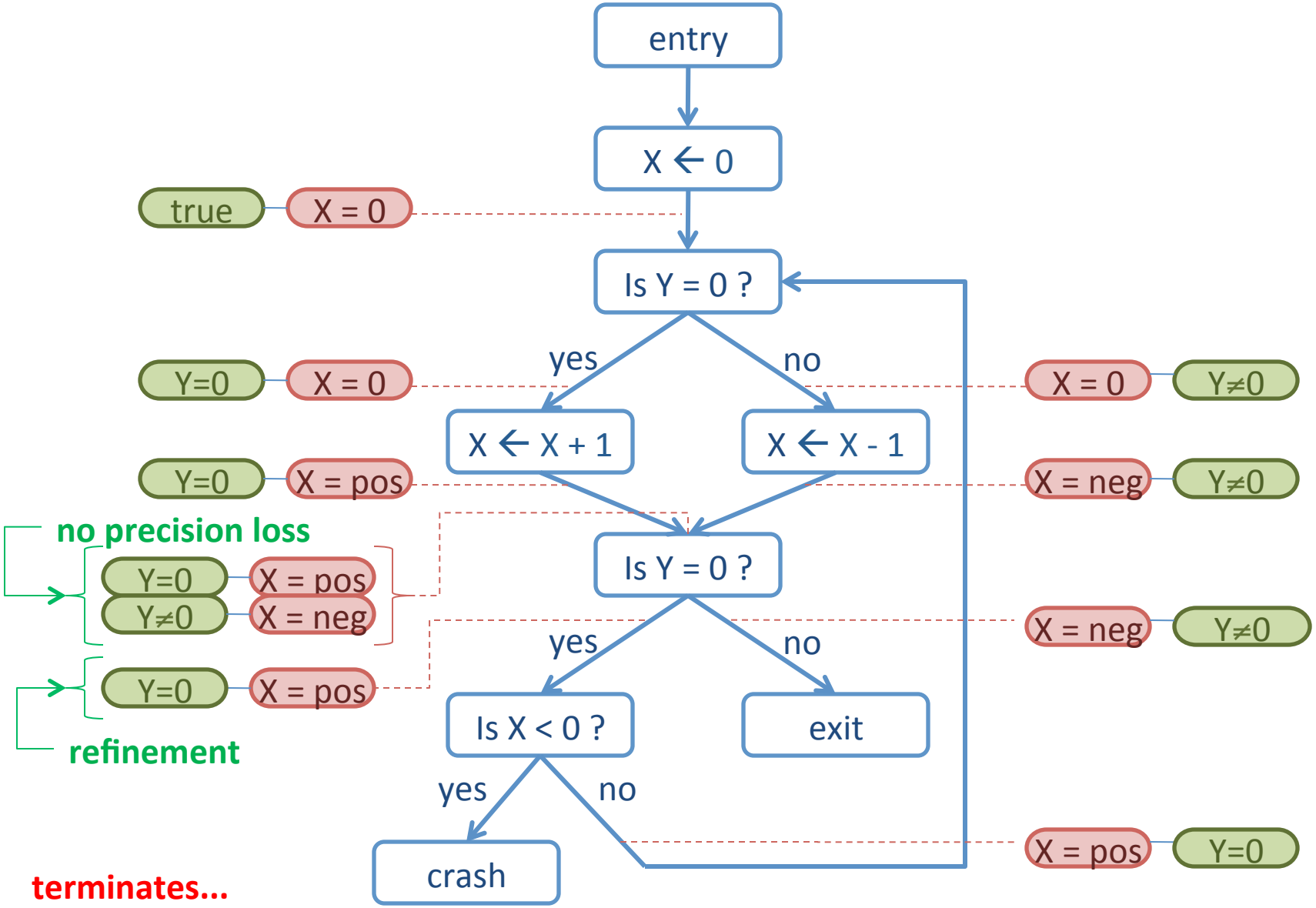
terminates…

… but reports false alarm

… therefore, need more precision


X = T
X ≠ neg
X = T
X ≠ pos
X = pos
X = 0
X = neg
X = ⊥
X = ⊥
true
Y = 0
Y ≠ 0
false
refinement lattice
sign lattice
Boolean formula lattice

Try analyzing with “path-sensitive signs” approximation…



terminates…

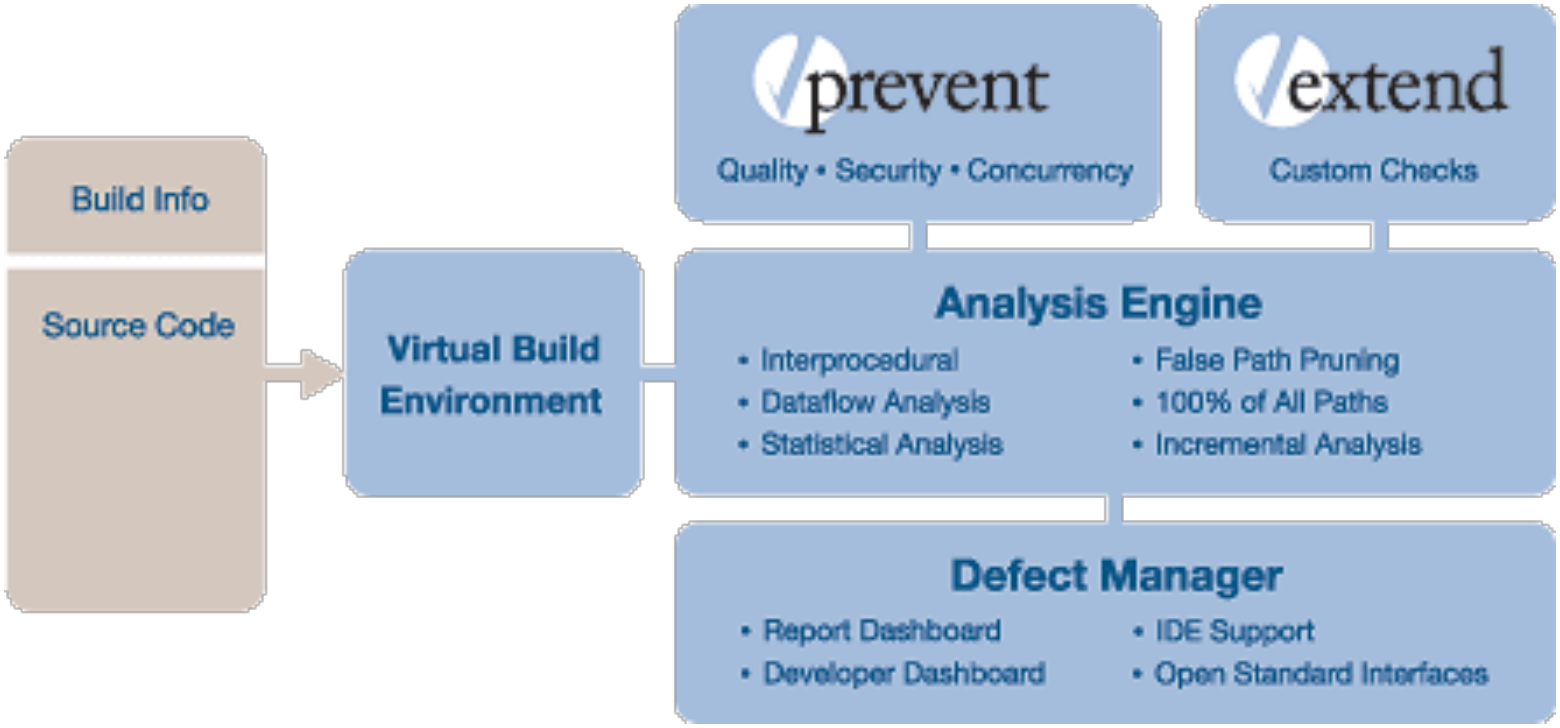
… no false alarm

… soundly proved never crashes

Outline

- General discussion of tools
  - Goals and limitations
  - Approach based on abstract states
- ★ More about one specific approach
  - Property checkers from Engler et al., Coverity
  - Sample security-related results
- Static analysis for Android malware
  - …

Slides from: S. Bugrahe, A. Chou, I&T Dillig, D. Engler, J. Franklin, A. Aiken, …


Build Info
Source Code
Virtual Build Environment
prevent
Quality • Security • Concurrency
extend
Custom Checks
Analysis Engine
• Interprocedural
• Dataflow Analysis
• Statistical Analysis
• False Path Pruning
• 100% of All Paths
• Incremental Analysis
Defect Manager
• Report Dashboard
• Developer Dashboard
• IDE Support
• Open Standard Interfaces

Bugs to Detect

Some examples

- Crash Causing Defects
- Null pointer dereference
- Use after free
- Double free
- Array indexing errors
- Mismatched array new/delete
- Potential stack overrun
- Potential heap overrun
- Return pointers to local variables
- Logically inconsistent code
- Uninitialized variables
- Invalid use of negative values
- Passing large parameters by value
- Underallocations of dynamic data
- Memory leaks
- File handle leaks
- Network resource leaks
- Unused values
- Unhandled return codes
- Use of invalid iterators

Slide credit: Andy Chou

Example: Check for missing optional args

- **Prototype for open() syscall:**

```
int open(const char *path, int oflag, /* mode_t mode */...);
```

- **Typical mistake:**

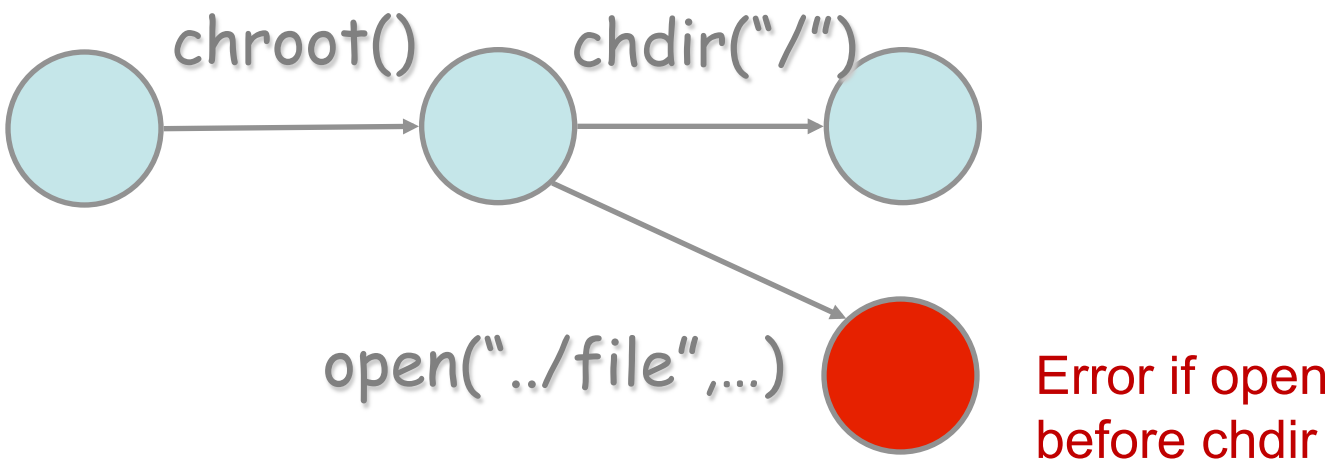
```
fd = open("file", O_CREAT);
```

- **Result: file has random permissions**

- **Check: Look for oflags == O_CREAT without mode argument**

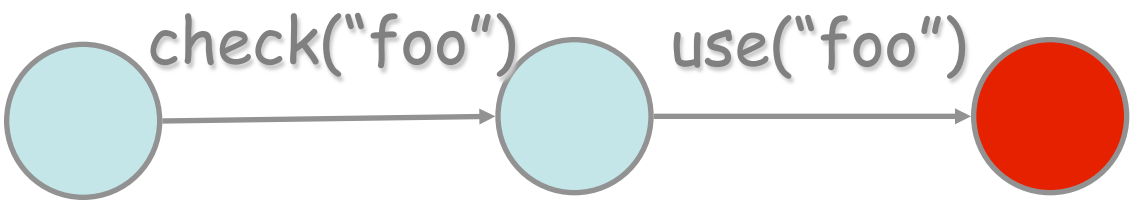
Example: Chroot protocol checker

- **Goal: confine process to a "jail" on the filesystem**
  - chroot() changes filesystem root for a process
- **Problem**
  - chroot() itself does not change current working directory

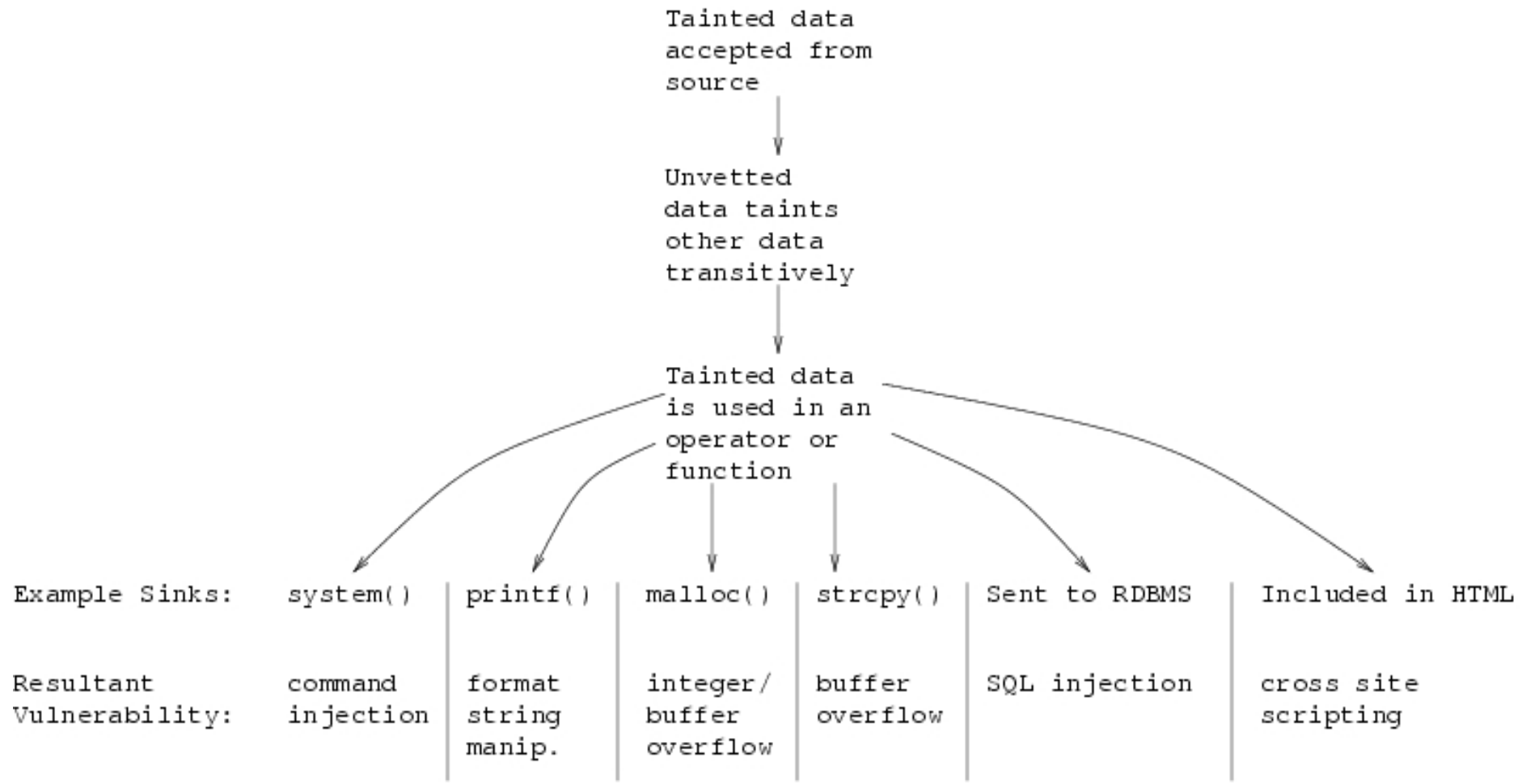


TOCTOU

- **Race condition between time of check and use**
- **Not applicable to all programs**



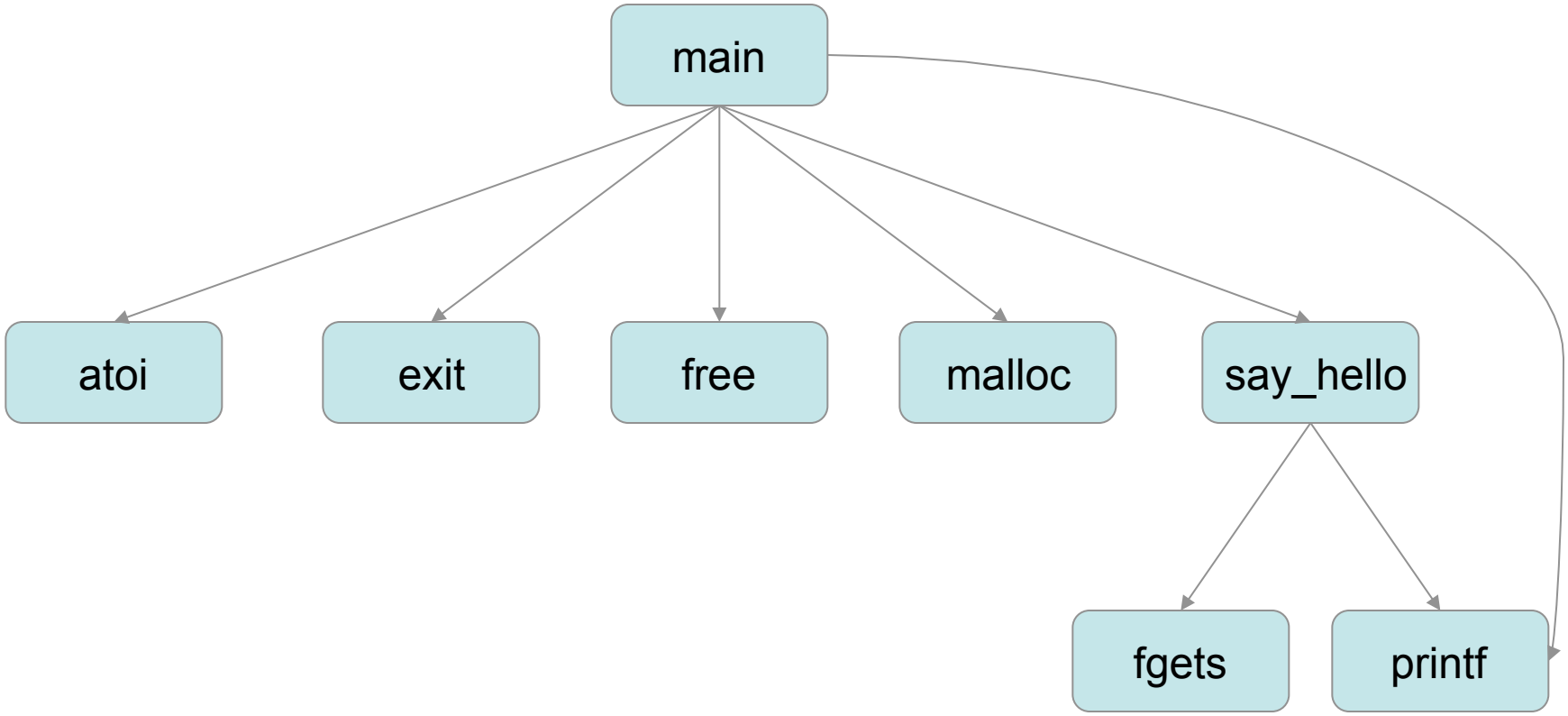
Tainting checkers



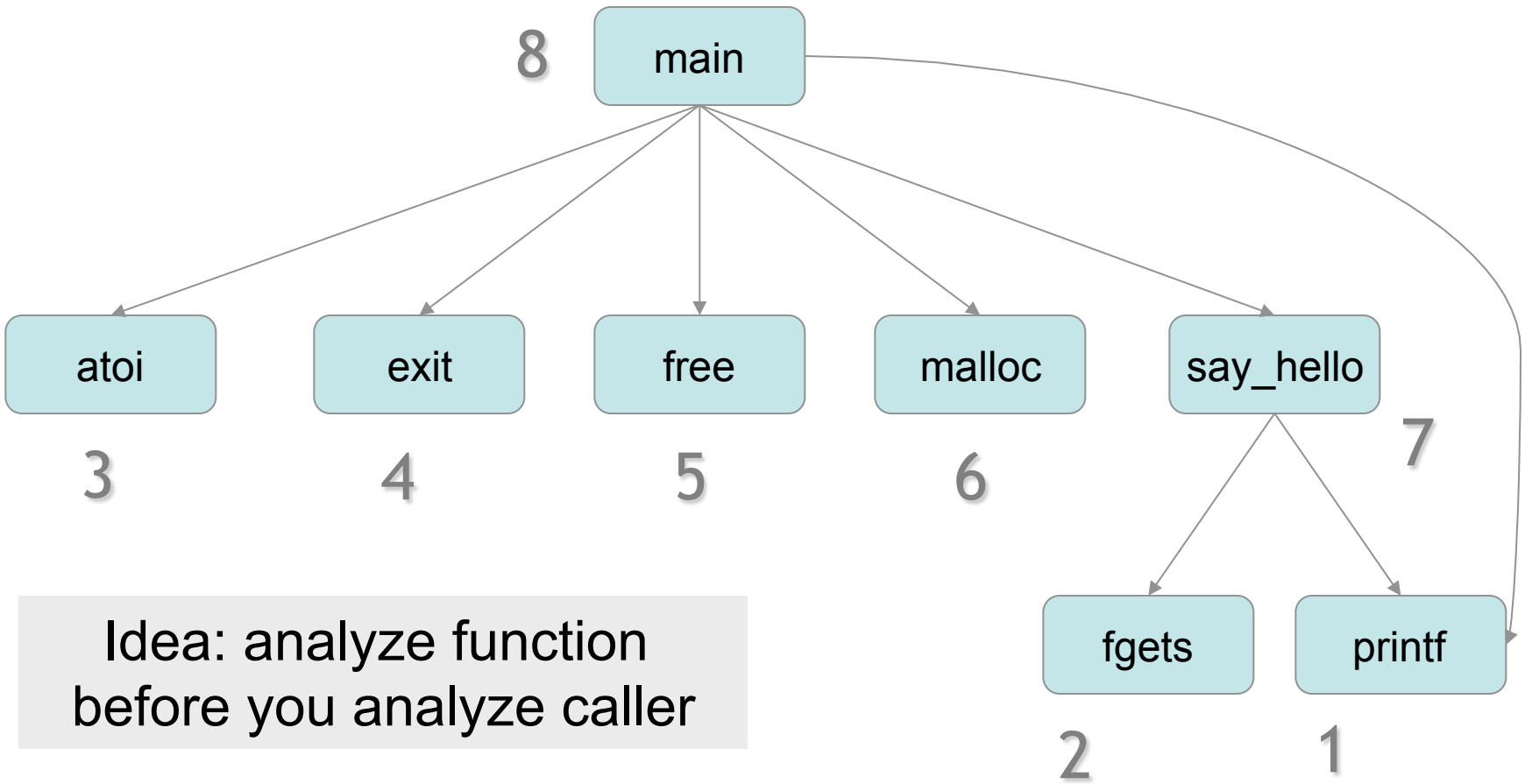
Example code with function def, calls

```
#include <stdlib.h>
#include <stdio.h>

void say_hello(char * name, int size) {
  printf("Enter your name: ");
  fgets(name, size, stdin);
  printf("Hello %s.\n", name);
}

int main(int argc, char *argv[]) {
  if (argc != 2) {
    printf("Error, must provide an input buffer size.\n");
    exit(-1);
  }
  int size = atoi(argv[1]);
  char * name = (char*)malloc(size);
  if (name) {
    say_hello(name, size);
    free(name);
  } else {
    printf("Failed to allocate %d bytes.\n", size);
  }
}
```

Callgraph

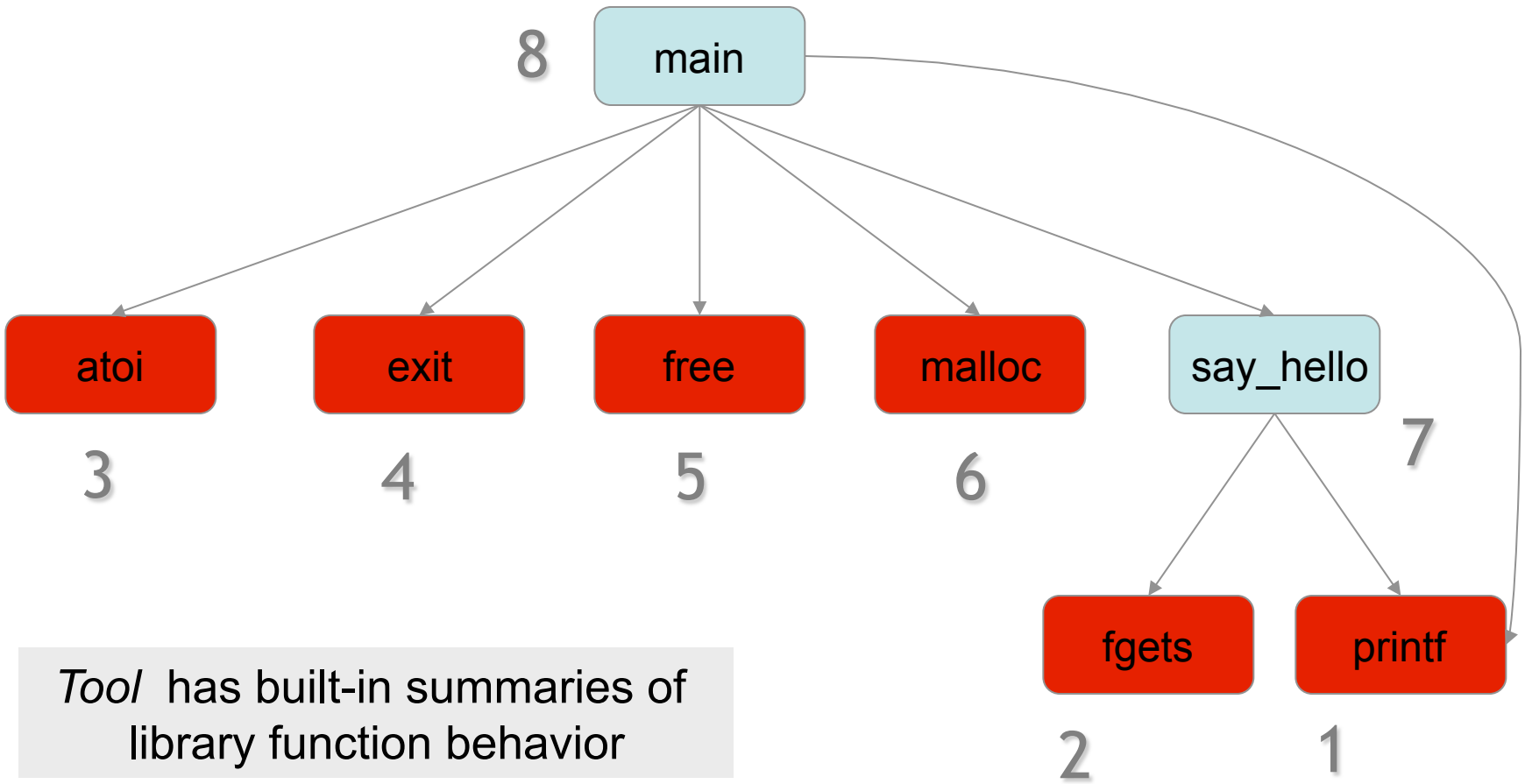


Reverse Topological Sort



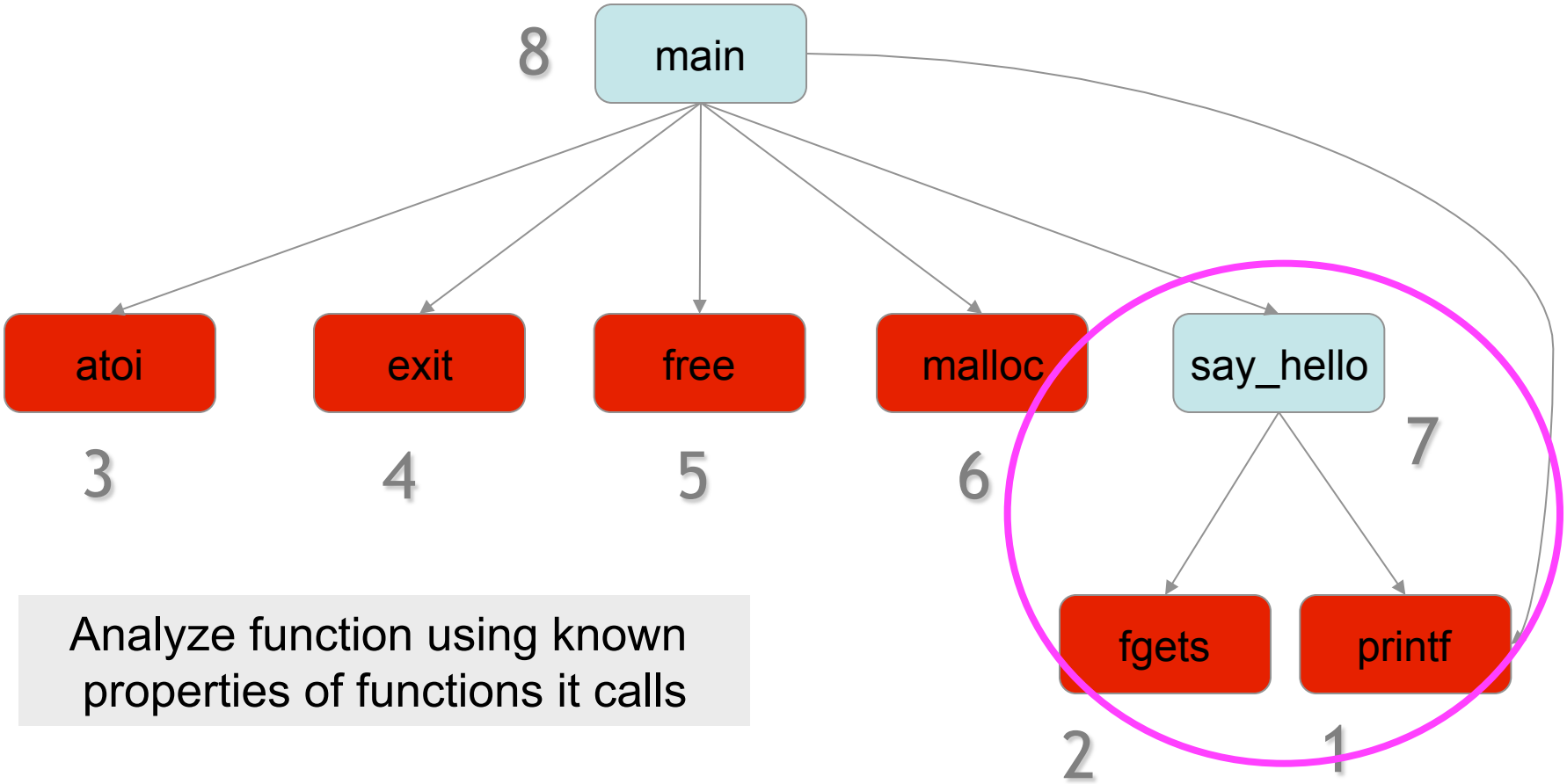
Idea: analyze function before you analyze caller

Apply Library Models



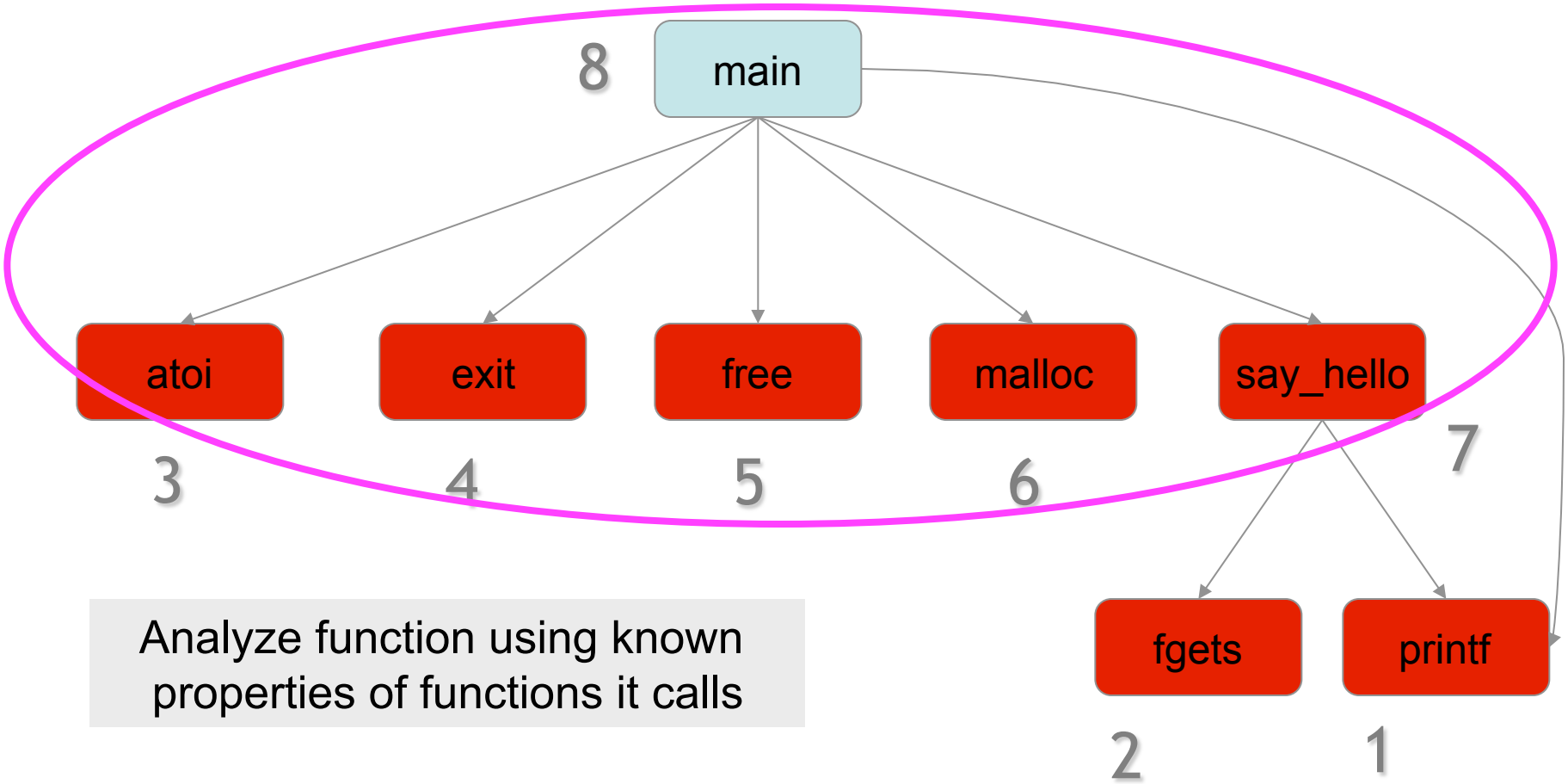
Tool has built-in summaries of library function behavior

Bottom Up Analysis



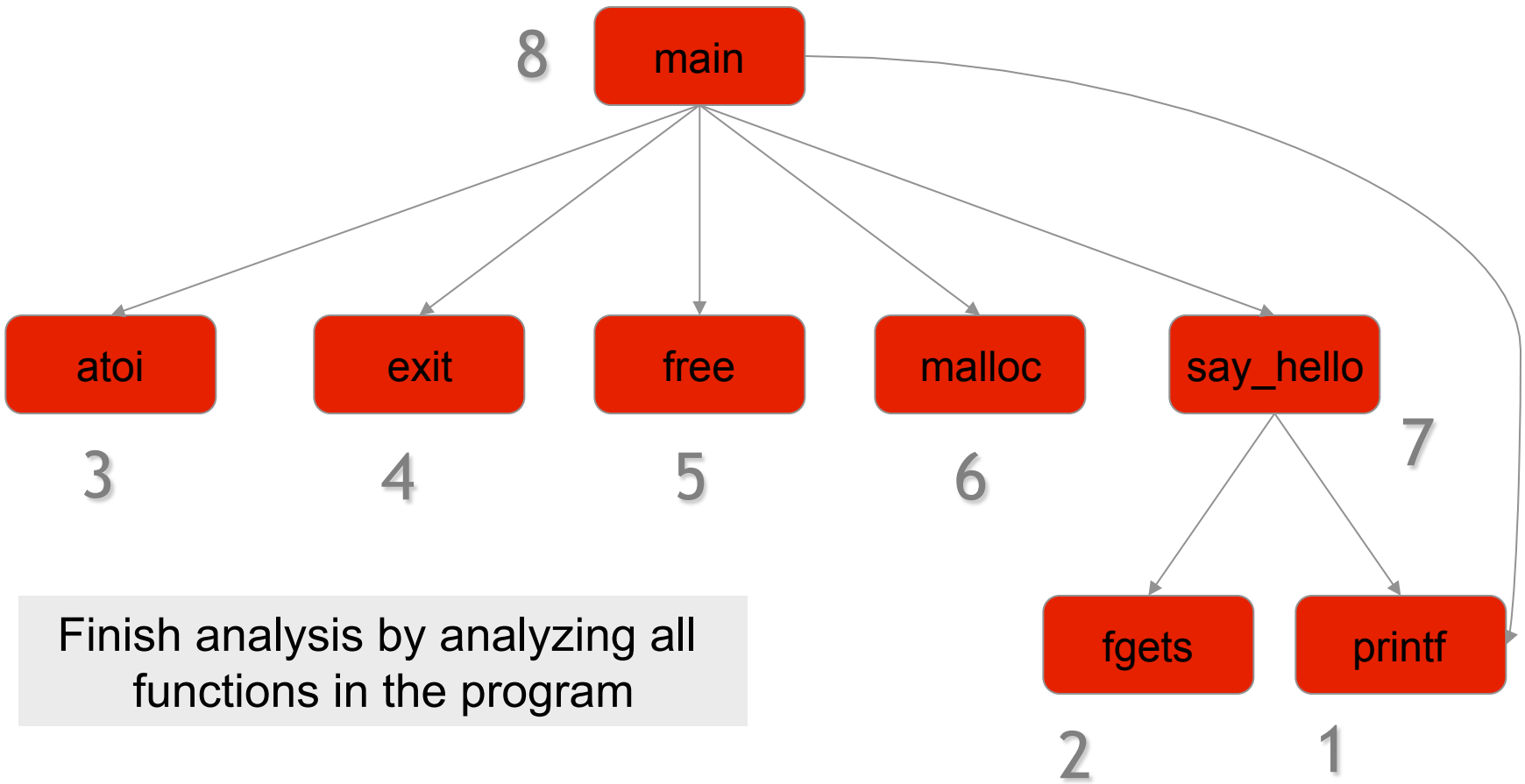
Analyze function using known properties of functions it calls

Bottom Up Analysis



Analyze function using known properties of functions it calls

Bottom Up Analysis

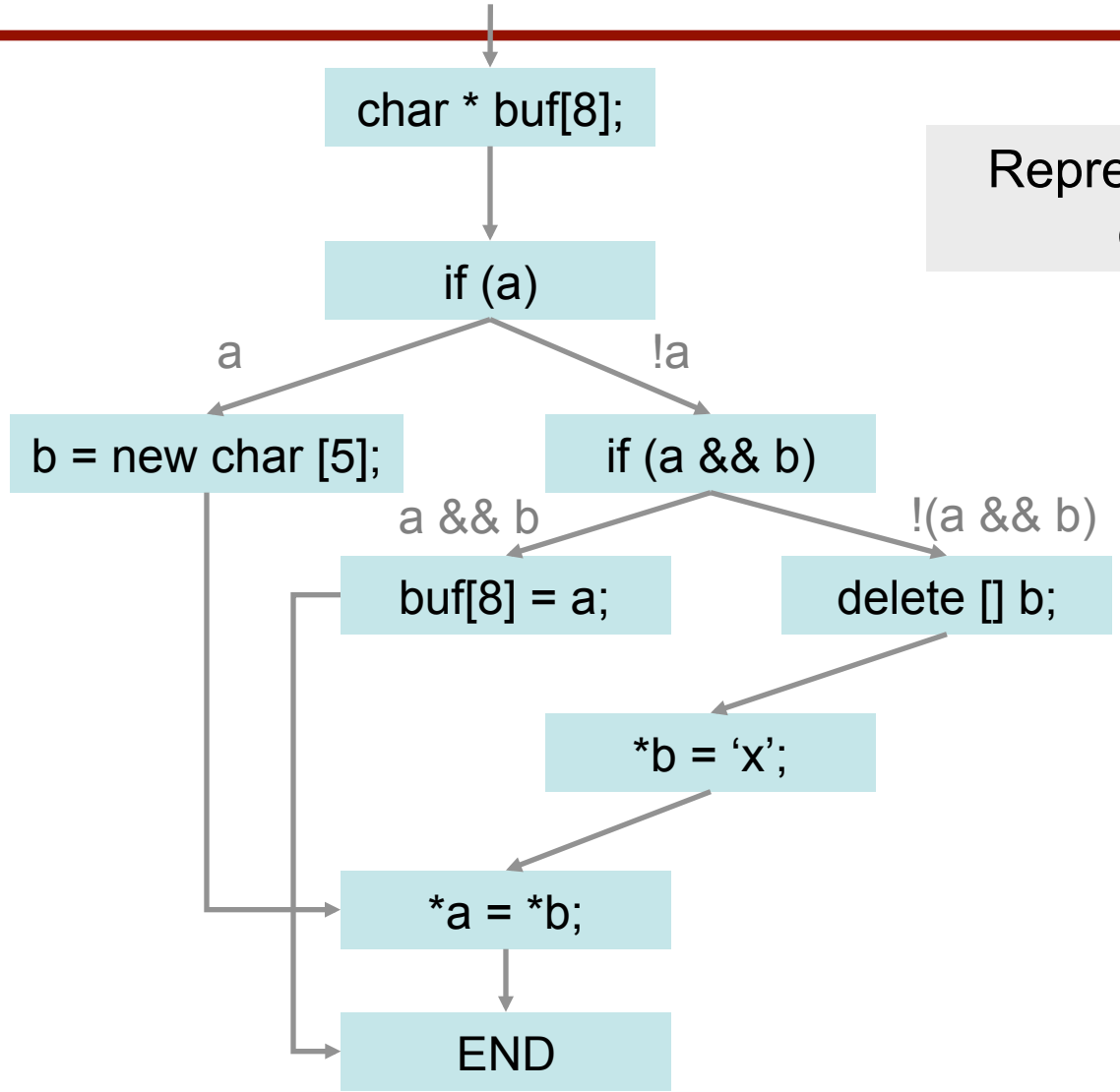


Finish analysis by analyzing all functions in the program

Finding Local Bugs

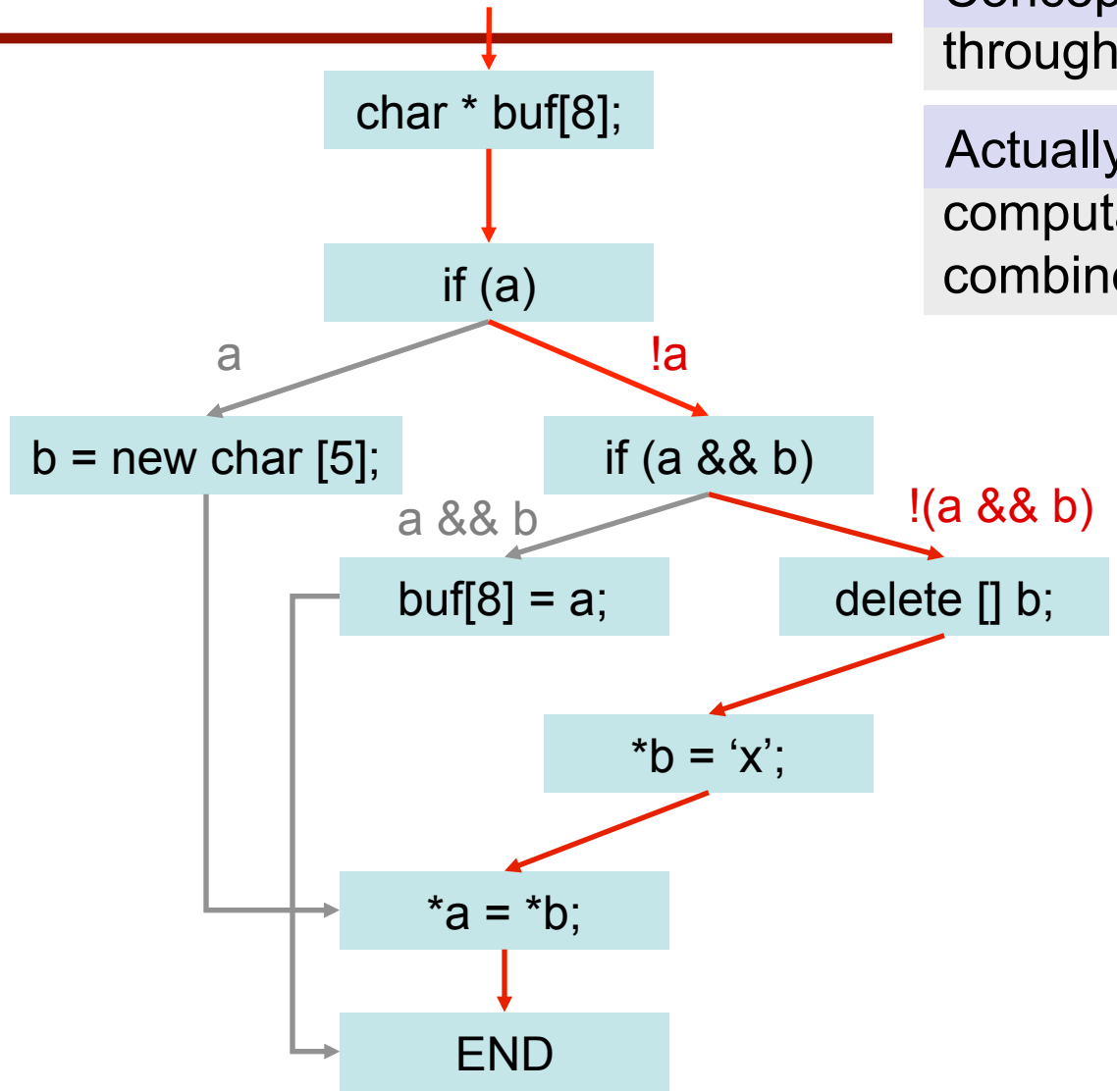
```
#define SIZE 8
void set_a_b(char * a, char * b) {
 char * buf[SIZE];
 if (a) {
     b = new char[5];
 } else {
     if (a && b) {
      buf[SIZE] = a;
      return;
     } else {
      delete [] b;
     }
     *b = 'x';
 }
 *a = *b;
}
```

Control Flow Graph



Represent logical structure of code in graph form

Path Traversal

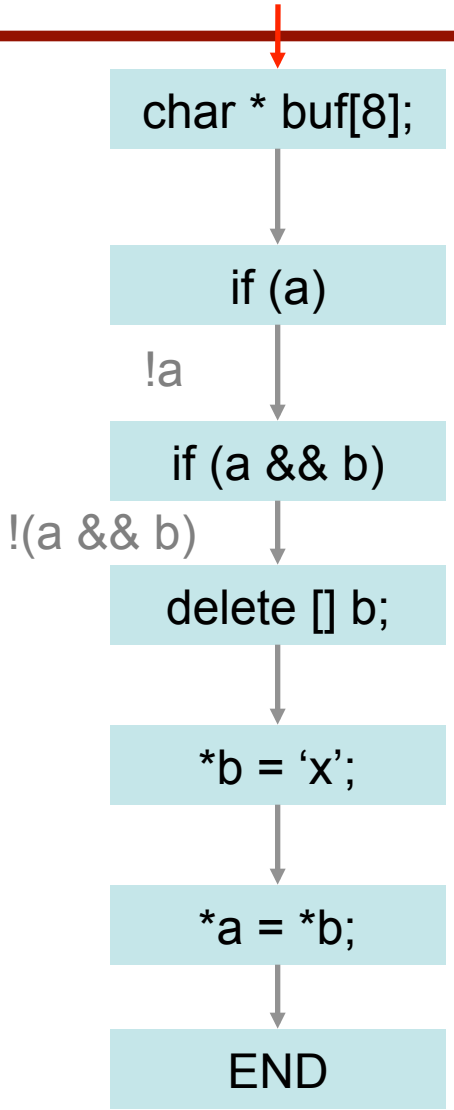


Conceptually Analyze each path through control graph separately

Actually Perform some checking computation once per node; combine paths at merge nodes

Apply Checking

Null pointers Use after free Array overrun



See how three checkers are run for this path

Checker

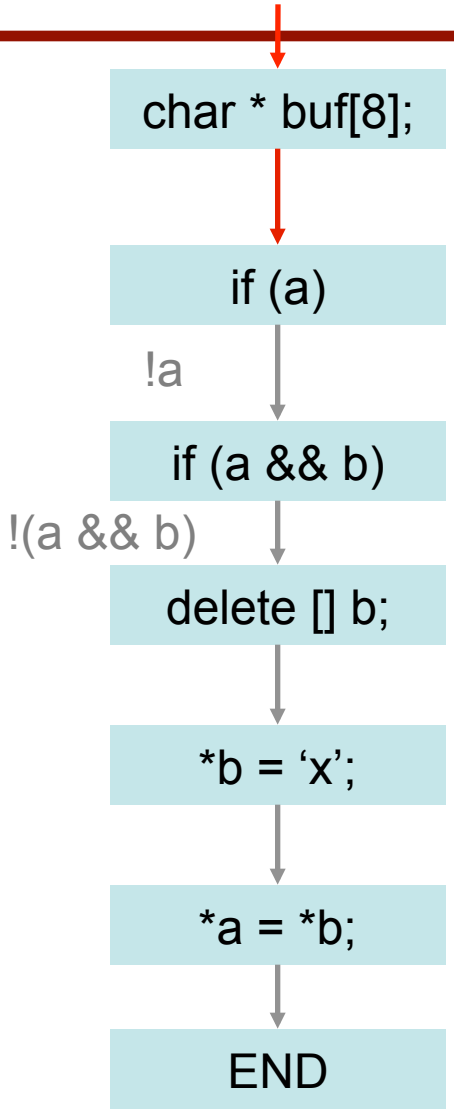
- Defined by a state diagram, with state transitions and error states

Run Checker

- Assign initial state to each program var
- State at program point depends on state at previous point, program actions
- Emit error if error state reached

Apply Checking

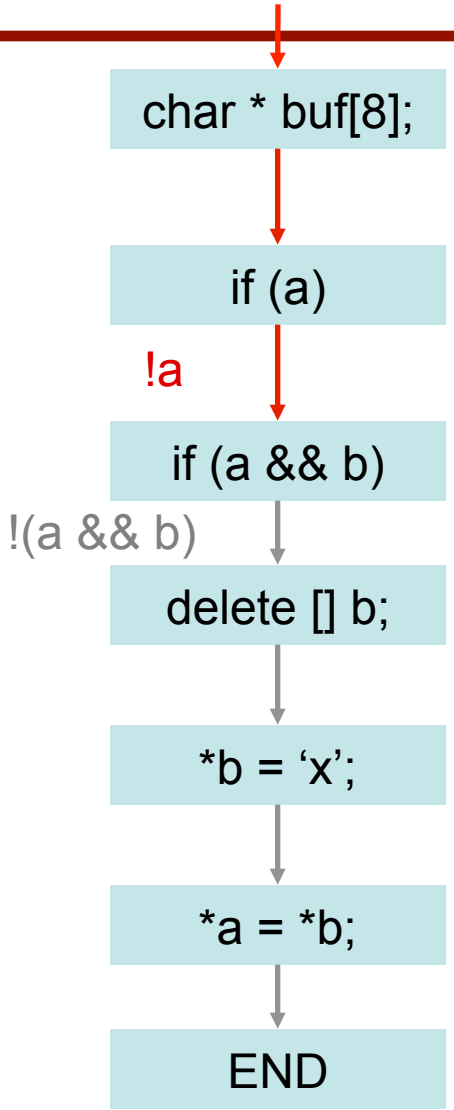
Null pointers Use after free Array overrun



"buf is 8 bytes"

Apply Checking

Null pointers Use after free Array overrun

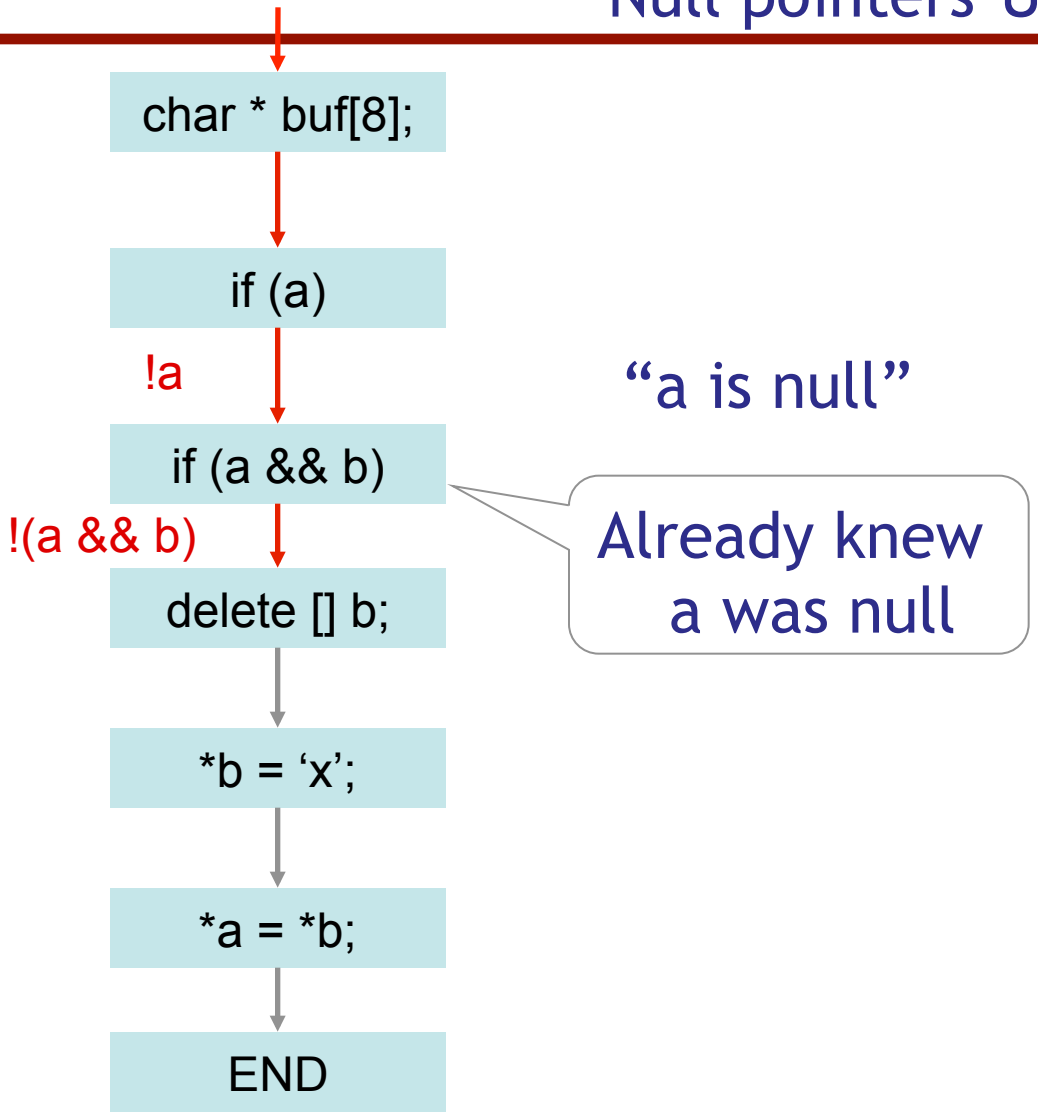


"buf is 8 bytes"

"a is null"

Apply Checking

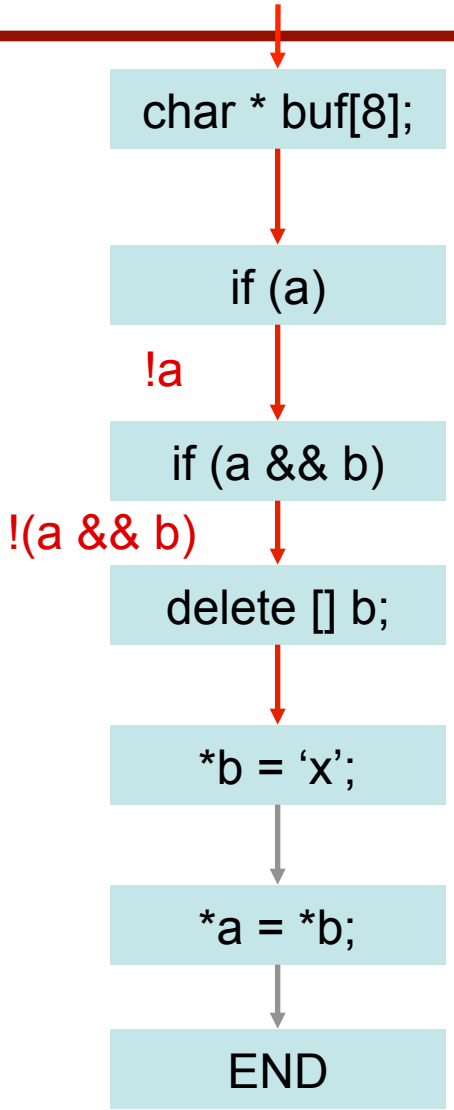
Null pointers Use after free Array overrun



"buf is 8 bytes"

Apply Checking

Null pointers Use after free Array overrun



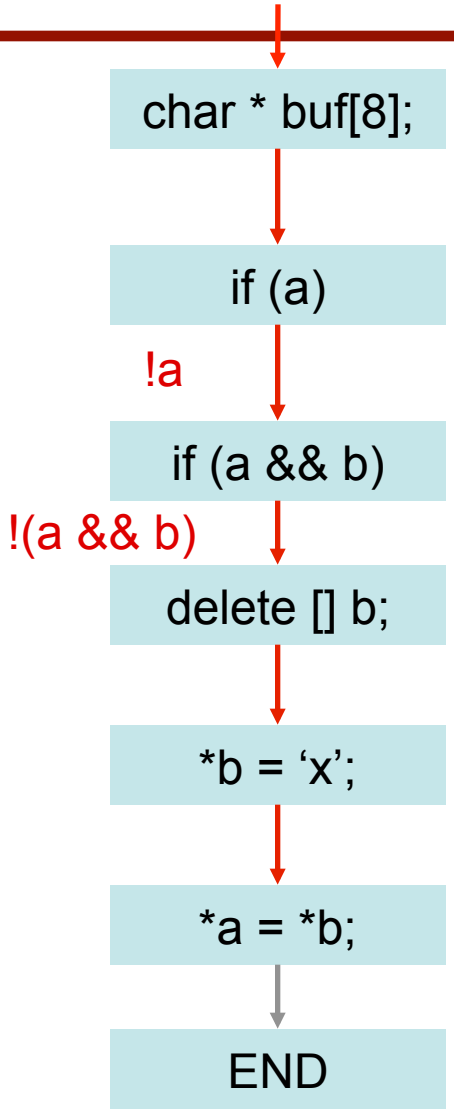
"buf is 8 bytes"

"a is null"

"b is deleted"

Apply Checking

Null pointers Use after free Array overrun



"buf is 8 bytes"

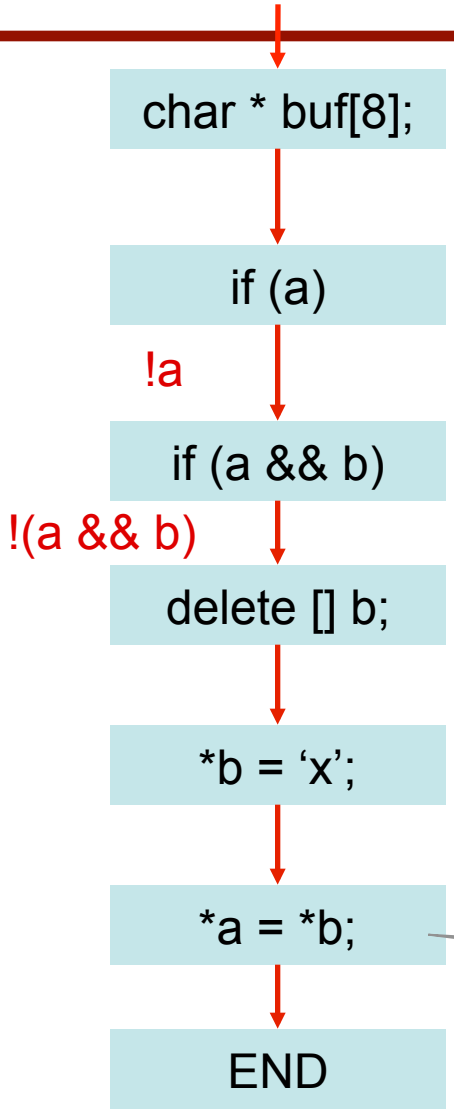
"a is null"

"b is deleted"

"b dereferenced!"

Apply Checking

Null pointers Use after free Array overrun



"buf is 8 bytes"

"a is null"

"b is deleted"

"b dereferenced!"

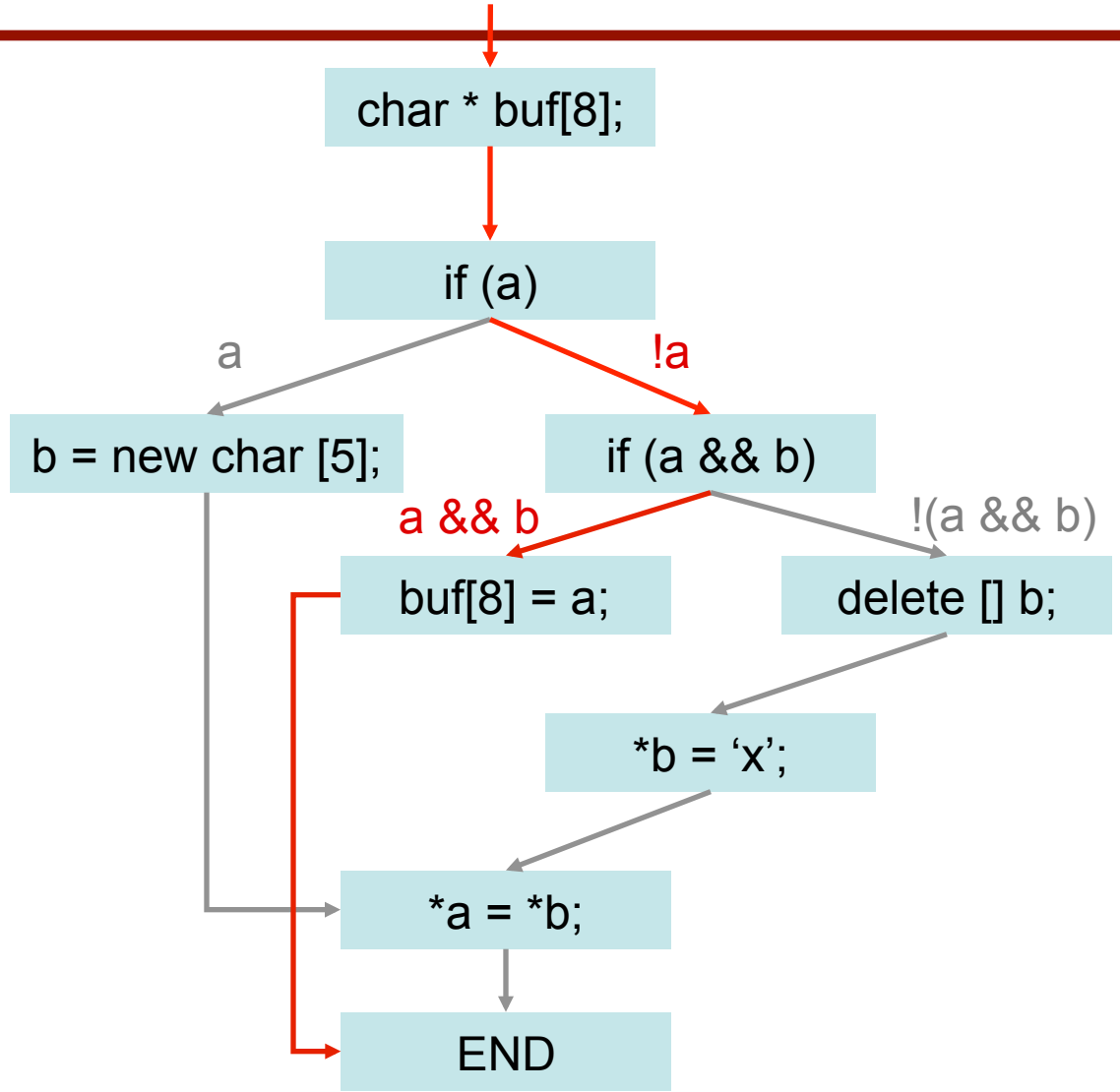
No more errors reported for b

False Positives

- **What is a bug? Something the user will fix.**

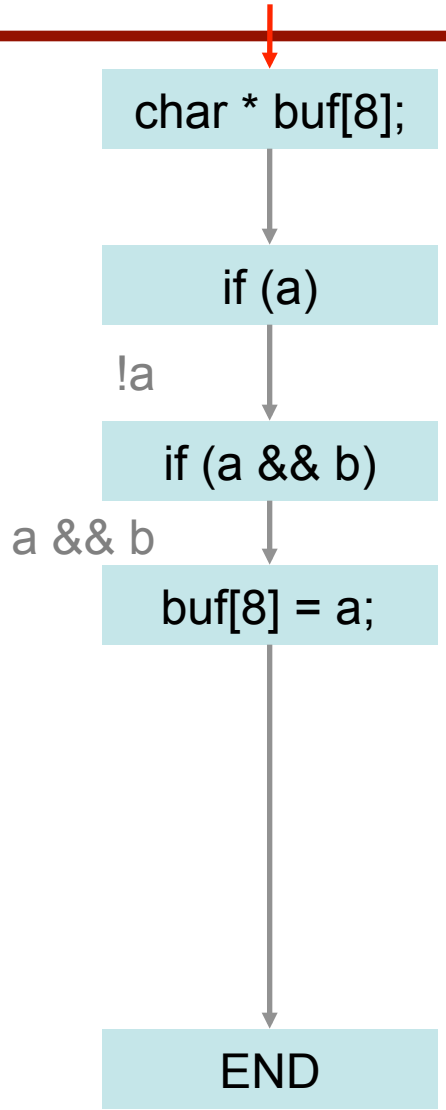
- **Many sources of false positives**
  - False paths
  - Idioms
  - Execution environment assumptions
  - Killpaths
  - Conditional compilation
  - “third party code”
  - Analysis imprecision
  - …

A False Path

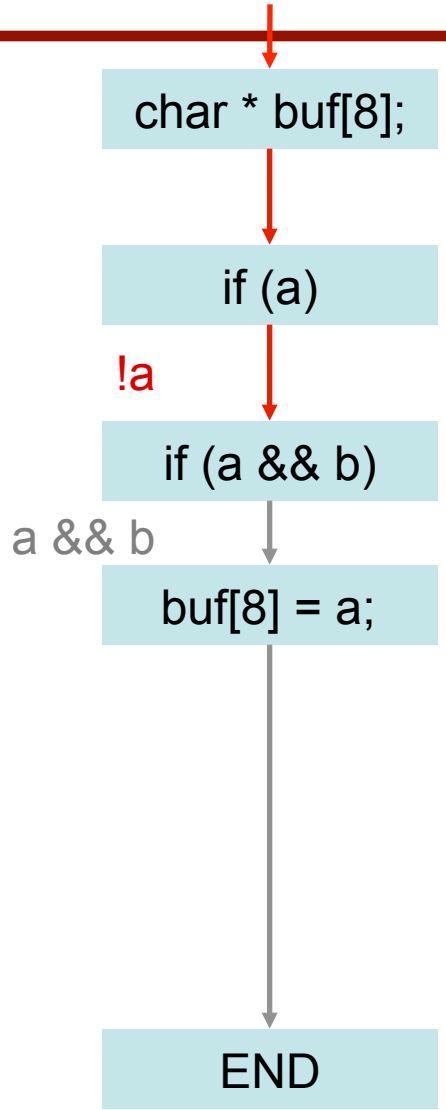


False Path Pruning

Integer Range Disequality Branch

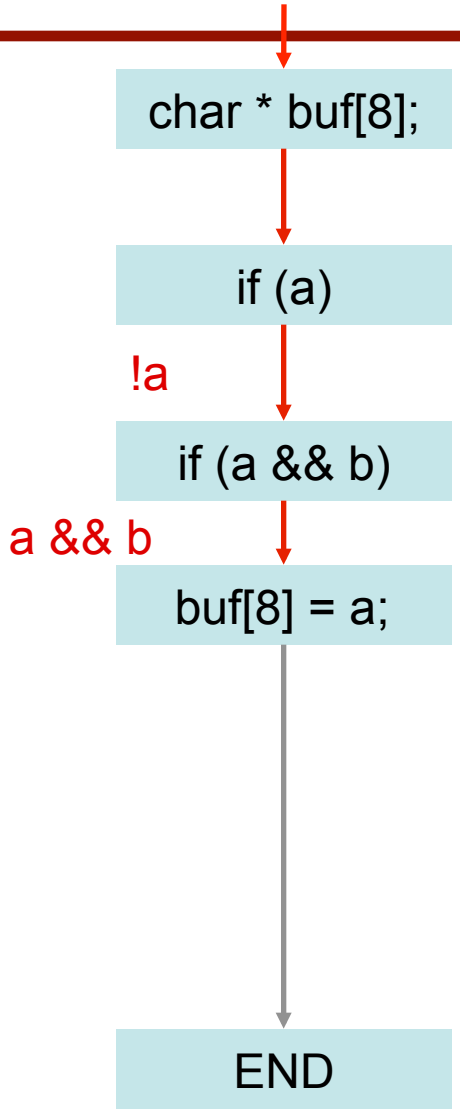


False Path Pruning



Integer Range	Disequality	Branch
“a in [0,0]”		“a == 0 is true”

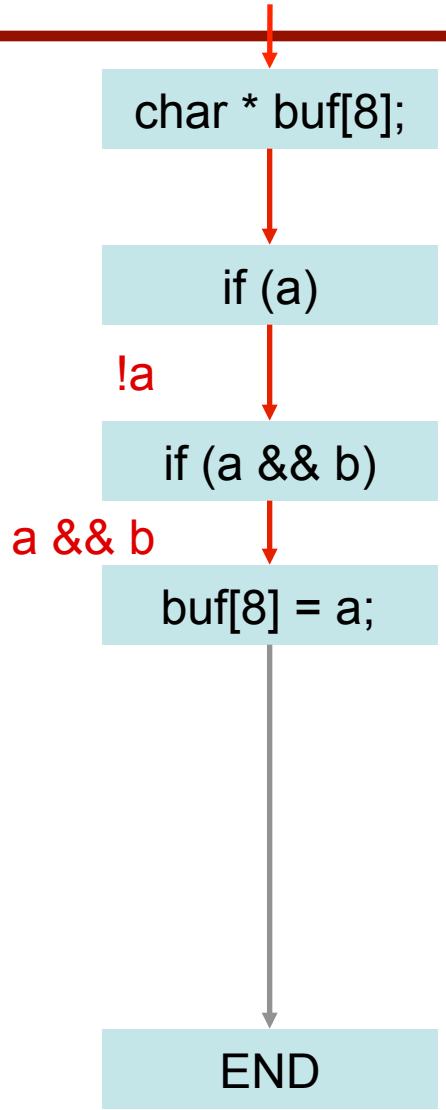
False Path Pruning



Integer Range	Disequality	Branch
“a in [0,0]”		“a == 0 is true”
	“a != 0”	

False Path Pruning

Integer Range Disequality Branch



Impossible

"a in [0,0]"

"a == 0 is true"

"a != 0"

Environment Assumptions

- **Should the return value of malloc() be checked?**

```
int *p = malloc(sizeof(int));
*p = 42;
```

OS Kernel:
Crash machine.

File server:
Pause filesystem.

Web application:
200ms downtime

Spreadsheet:
Lose unsaved changes.

Game:
Annoy user.

IP Phone:
Annoy user.

Library:
?

Medical device:
malloc?!

Statistical Analysis

- **Assume the code is usually right**

3/4 deref

```
int *p = malloc(sizeof(int));
*p = 42;
```

```
int *p = malloc(sizeof(int));
*p = 42;
```

```
int *p = malloc(sizeof(int));
*p = 42;
```

```
int *p = malloc(sizeof(int));
if(p)  *p = 42;
```

1/4 deref

```
int *p = malloc(sizeof(int));
if(p) *p = 42;
```

```
int *p = malloc(sizeof(int));
if(p) *p = 42;
```

```
int *p = malloc(sizeof(int));
if(p) *p = 42;
```

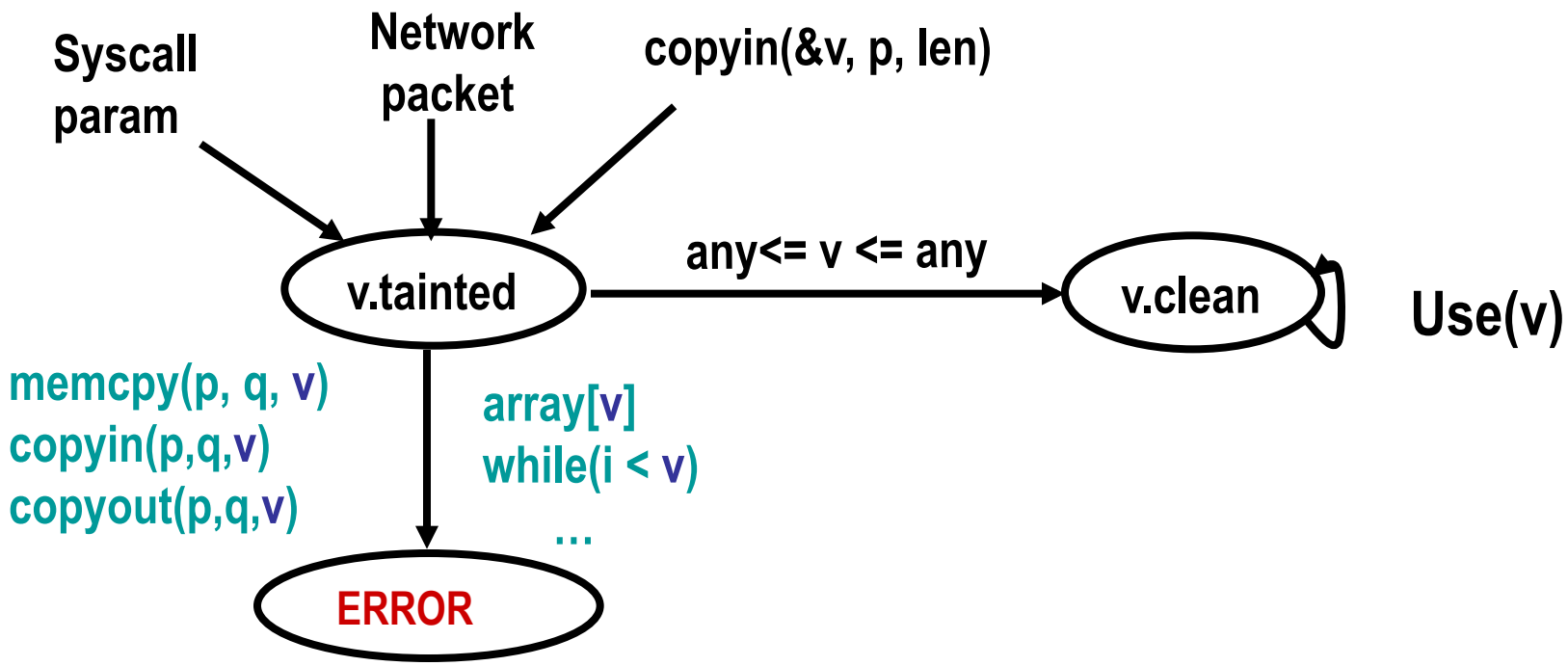
```
int *p = malloc(sizeof(int));
*p = 42;
```

Application to Security Bugs

- **Stanford research project**
  - Ken Ashcraft and Dawson Engler, Using Programmer-Written Compiler Extensions to Catch Security Holes, IEEE Security and Privacy 2002
  - Used modified compiler to find over 100 security holes in Linux and BSD
  - http://www.stanford.edu/~engler/
- **Benefit**
  - Capture recommended practices, known to experts, in tool available to all

Sanitize integers before use

Warn when unchecked integers from untrusted sources reach trusting sinks



Linux: 125 errors, 24 false; BSD: 12 errors, 4 false

Example security holes

- **Remote exploit, no checks**

```
/* 2.4.9/drivers/isdn/act2000/capi.c:actcapi_dispatch */
isdn_ctrl cmd;
...
while ((skb = skb_dequeue(&card->rcvq))) {
   msg = skb->data;
   ...
   memcpy(cmd.parm.setup.phone,
          msg->msg.connect_ind.addr.num,
          msg->msg.connect_ind.addr.len - 1);
```

Example security holes

- **Missed lower-bound check:**

```
/* 2.4.5/drivers/char/drm/i810_dma.c */

if(copy_from_user(&d, arg, sizeof(arg)))
   return -EFAULT;
if(d.idx > dma->buf_count)
   return -EINVAL;
buf = dma->buflist[d.idx];
Copy_from_user(buf_priv->virtual, d.address, d.used);
```

User-pointer inference

- **Problem: which are the user pointers?**
  - Hard to determine by dataflow analysis
  - Easy to tell if kernel *believes* pointer is from user!
- **Belief inference**
  - "*p" implies safe kernel pointer
  - "copyin(p)/copyout(p)" implies dangerous user ptr
  - Error: pointer p has both beliefs.
- **Implementation: 2 pass checker**

  inter-procedural: compute all tainted pointers

  local pass to check that they are not dereferenced

Results for BSD and Linux

- **All bugs released to implementers; most serious fixed**

Violation	Linux Bug	Linux Fixed	BSD Bug	BSD Fixed
Gain control of system	18	15	3	3
Corrupt memory	43	17	2	2
Read arbitrary memory	19	14	7	7
Denial of service	17	5	0	0
Minor	28	1	0	0
Total	125	52	12	12

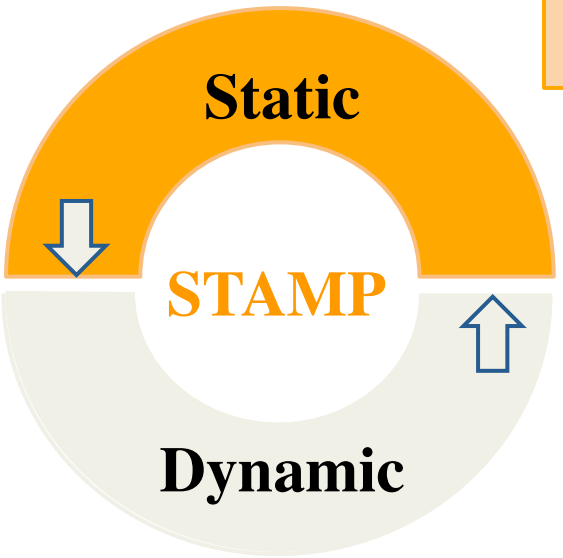
Outline

- General discussion of tools
  - Goals and limitations
  - Approach based on abstract states
- More about one specific approach
  - Property checkers from Engler et al., Coverity
  - Sample security-related results
- ★ Static analysis for Android malware
  - …

Slides from: S. Bugrahe, A. Chou, I&T Dillig, D. Engler, J. Franklin, A. Aiken, …

STAMP Admission System

Static Analysis
More behaviors,
fewer details



Dynamic Analysis
Fewer behaviors,
more details

Alex Aiken,
John Mitchell,
Saswat Anand,
Jason Franklin
Osbert Bastani,
Lazaro Clapp,
Patrick Mutchler,
Manolis Papadakis

Analysis

Step 1

Convert bytecode to intermediate format (called Quads)

Step 2

Compute call graph using Class Hierarchy Analysis

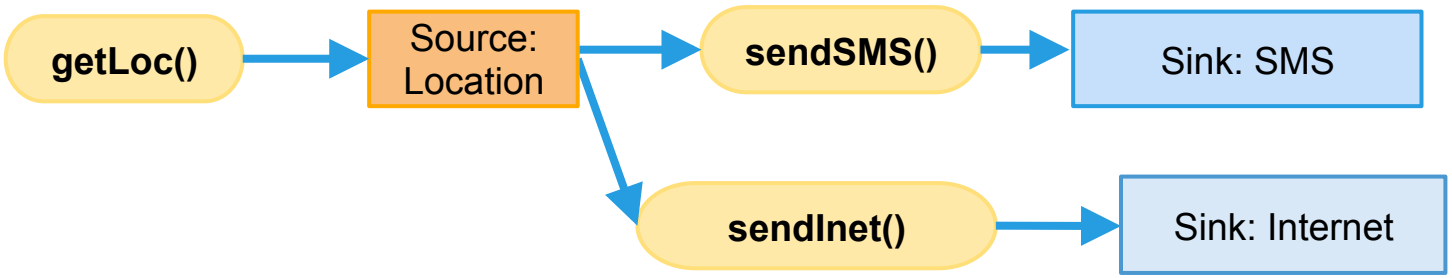
Step 3

Build an edge-labeled graph G by processing Quads of each class

Step 4

Add new edges to G as per a set of rules until no rules apply

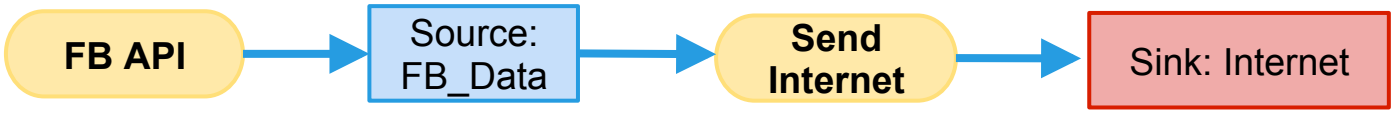
Data Flow Analysis



- Source-to-sink flows
  - Sources: Location, Calendar, Contacts, Device ID etc.
  - Sinks: Internet, SMS, Disk, etc.

Data Flow Analysis in Action

- Malware/Greyware Analysis
  - Data flow summaries enable enterprise-specific policies
- API Misuse and Data Theft Detection



- Automatic Generation of App Privacy Policies
  - Avoid liability, protect consumer privacy

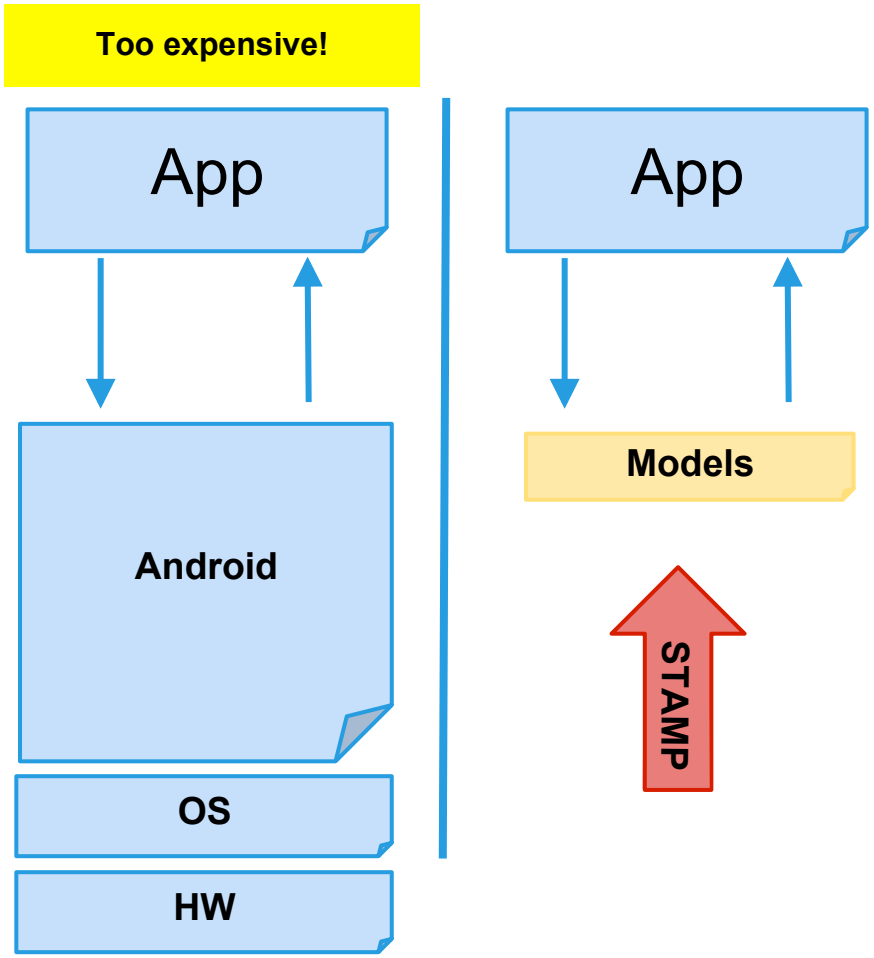
Privacy Policy
This app collects your:
Contacts
Phone Number
Address

- Vulnerability Discovery

Challenges

- Android is 3.4M+ lines of complex code
  - Uses reflection, callbacks, native code
- **Scalability:** Whole system analysis impractical
- **Soundness:** Avoid missing flows
- **Precision:** Minimize false positives

STAMP Approach



- Model Android/Java
  - Sources and sinks
  - Data structures
  - Callbacks
  - 500+ models
- Whole-program analysis
  - Context sensitive

Building Models

- 30k+ methods in Java/Android API
  - 5 mins x 30k = 2500 hours

- Follow the permissions
  - 20 permissions for sensitive sources
    - ACCESS_FINE_LOCATION (8 methods with source annotations)
    - READ_PHONE_STATE - (9 methods)
  - 4 permissions for sensitive sinks
    - INTERNET, SEND_SMS, etc.

Identifying Sensitive Data

```
android.Telephony.TelephonyManager: String getDeviceId()
```

- Returns device IMEI in String
- Requires permission GET_PHONE_STATE

```
@STAMP(
   SRC ="$GET_PHONE_STATE.deviceid",
   SINK ="@return"
)
```

Data We Track (Sources)

- Account data
- Audio
- Calendar
- Call log
- Camera
- Contacts
- Device Id
- Location
- Photos (Geotags)
- SD card data
- SMS

30+ types of sensitive data

Data Destinations (Sinks)

- Internet (socket)
- SMS
- Email
- System Logs
- Webview/Browser
- File System
- Broadcast Message

10+ types of exit points

Currently Detectable Flow Types

396 Flow Types

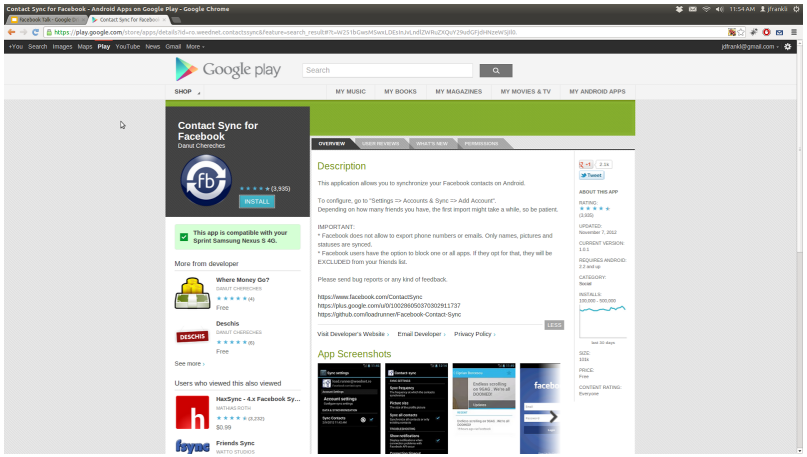
Unique Flow Types = Sources x Sink

Example Analysis

Contact Sync for Facebook (unofficial)

Description:

This application allows you to synchronize your Facebook contacts on Android.



IMPORTANT:

* "Facebook does not allow [sic] to export phone numbers or emails. Only names, pictures and statuses are synced."

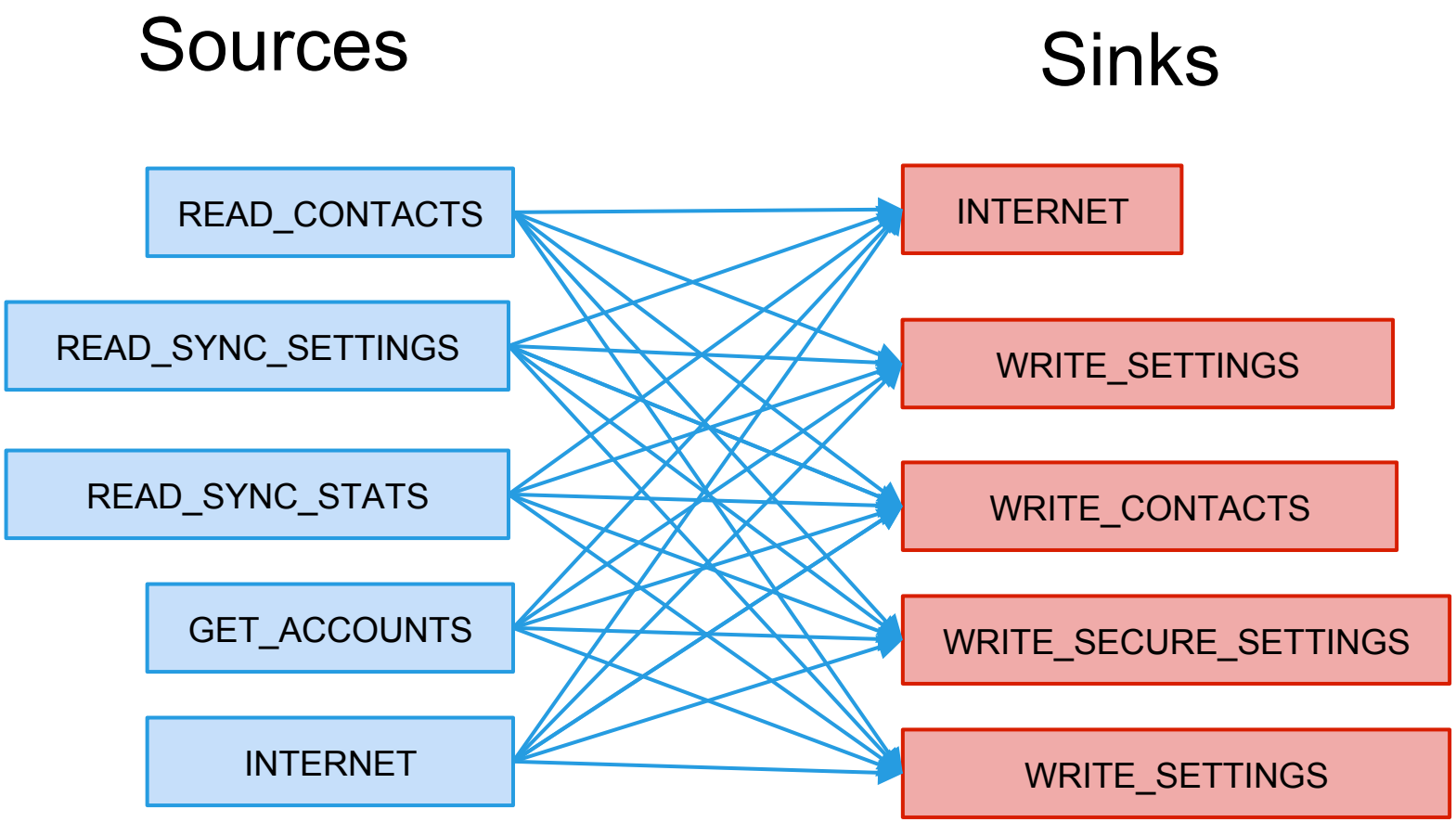
* "Facebook users have the option to block one or all apps. If they opt for that, they will be EXCLUDED from your friends list."

Privacy Policy: (page not found)

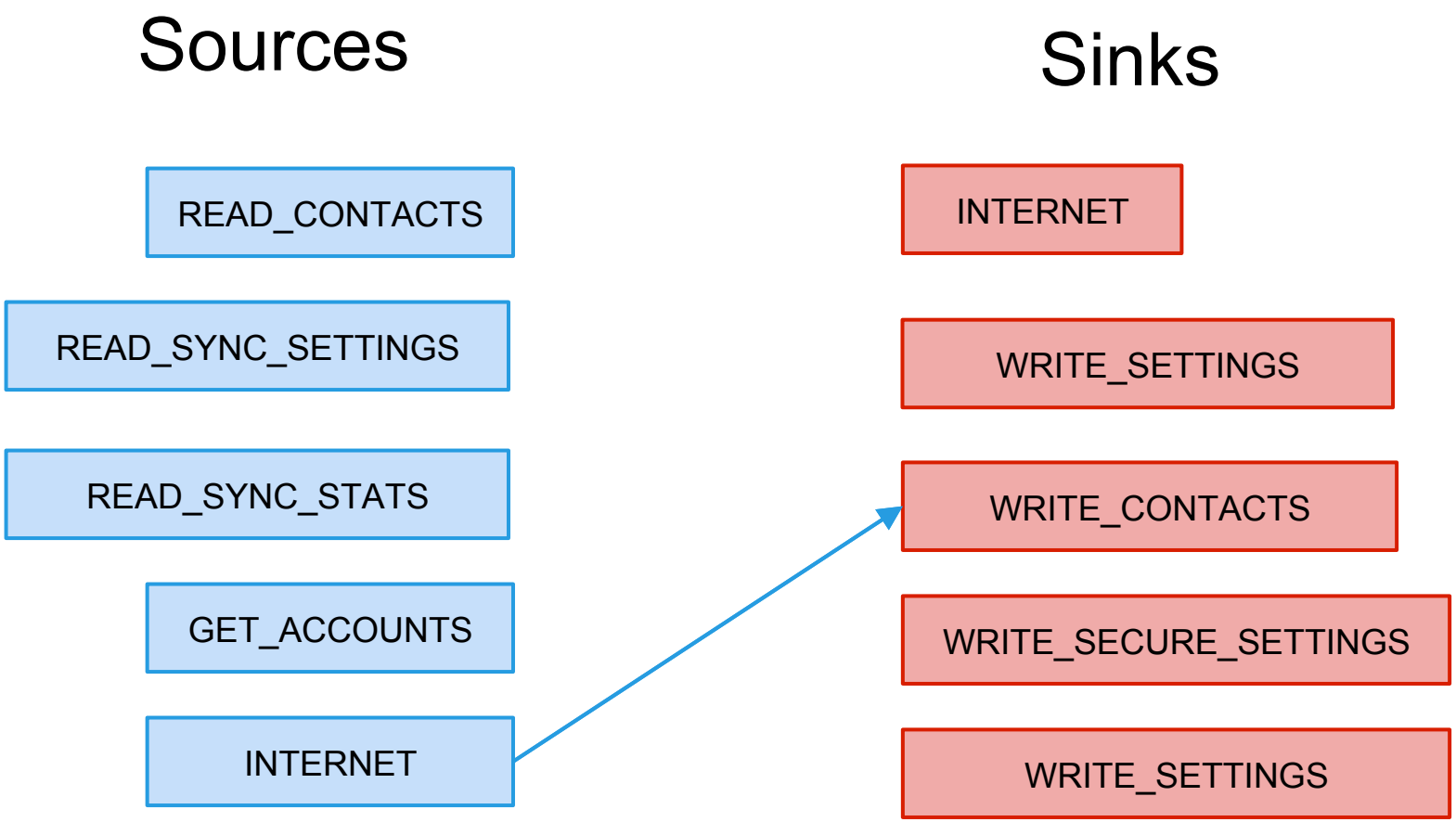
Contact Sync Permissions

Category	Permission	Description
Your Accounts	AUTHENTICATE_ACCOUNTS	Act as an account authenticator
	MANAGE_ACCOUNTS	Manage accounts list
	USE_CREDENTIALS	Use authentication credentials
Network Communication	**INTERNET**	**Full Internet access**
	ACCESS_NETWORK_STATE	View network state
Your Personal Information	**READ_CONTACTS**	**Read contact data**
	WRITE_CONTACTS	Write contact data
System Tools	WRITE_SETTINGS	Modify global system settings
	WRITE_SYNC_SETTINGS	Write sync settings (e.g. Contact sync)
	READ_SYNC_SETTINGS	Read whether sync is enabled
	READ_SYNC_STATS	Read history of syncs
Your Accounts	GET_ACCOUNTS	Discover known accounts
Extra/Custom	WRITE_SECURE_SETTINGS	Modify secure system settings

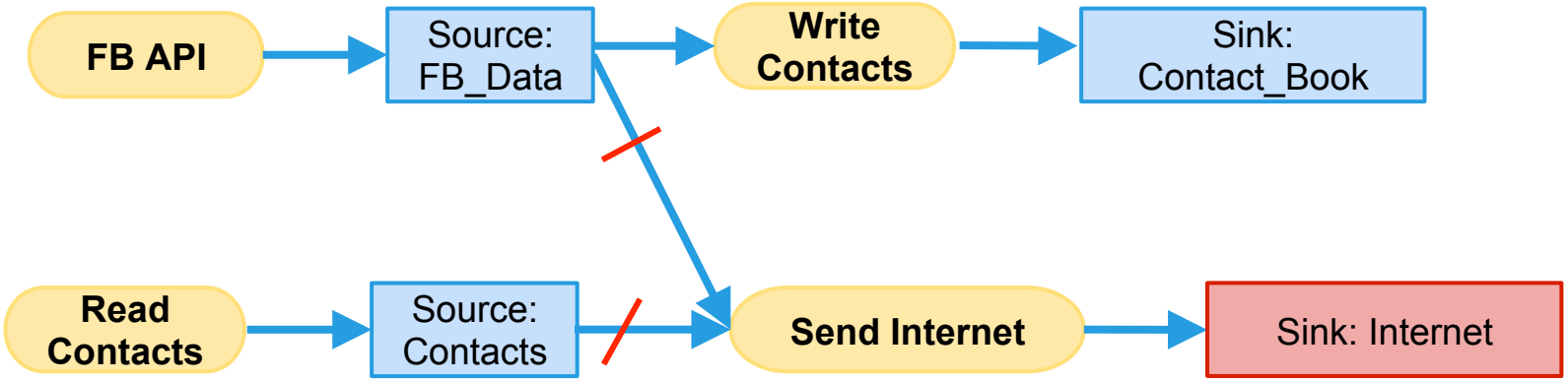
Possible Flows from Permissions



Expected Flows



Observed Flows



Example Study: Mobile Web Apps

- Goal

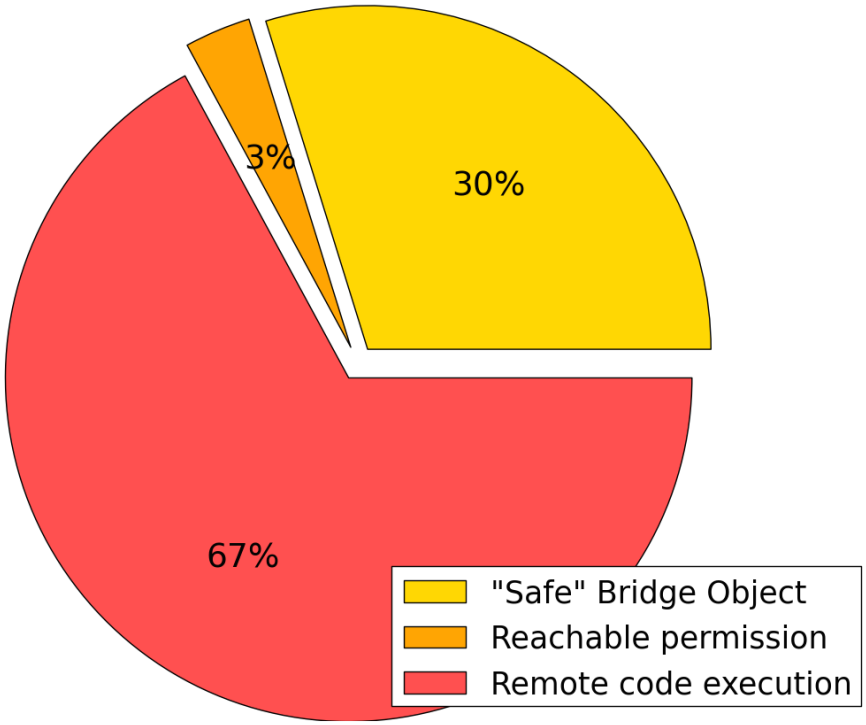
  Identify security concerns and vulnerabilities specific to mobile apps that access the web using an embedded browser

- Technical summary
  - WebView object renders web content
  - methods loadUrl, loadData, loadDataWithBaseUrl, postUrl
  - addJavascriptInterface(obj, name) allows JavaScript code in the web content to call Java object method name.foo()

Summary

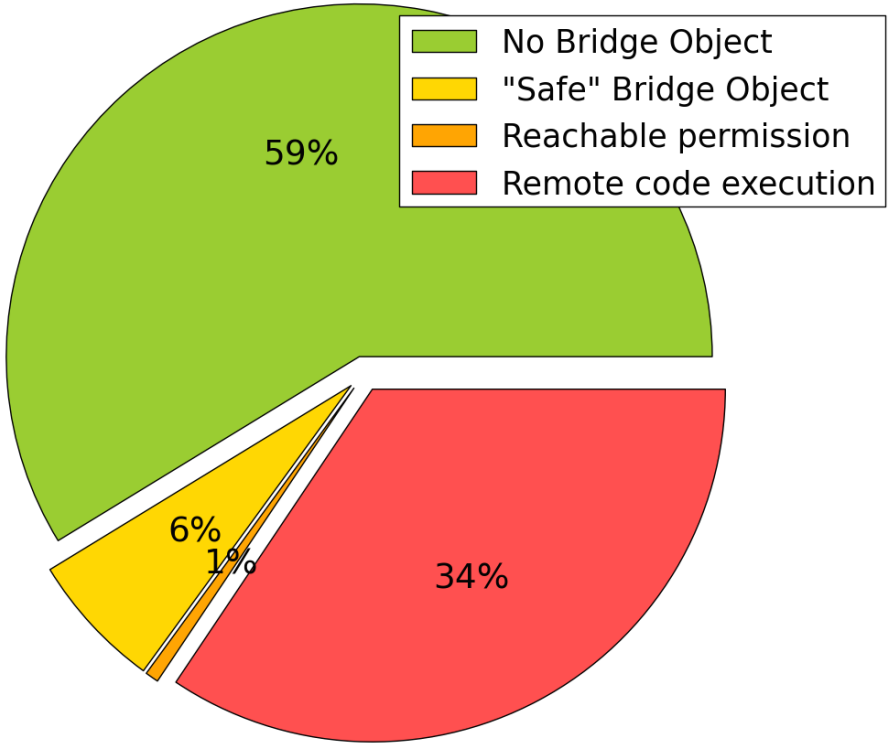
- Analyze a dataset of 737,828 Android apps
- Found large number of apps contain severe vulnerabilities
- 37,418 apps are vulnerable to a remote code execution exploit when run on any Android device, because of security oversight in older versions and slow adoption of safe versions
- 45,689 apps are vulnerable to a remote code execution exploit when run on 73% of the in-use Android devices.
- Offer recommendations for developers who wish to avoid these vulnerabilities.

Origin Severity



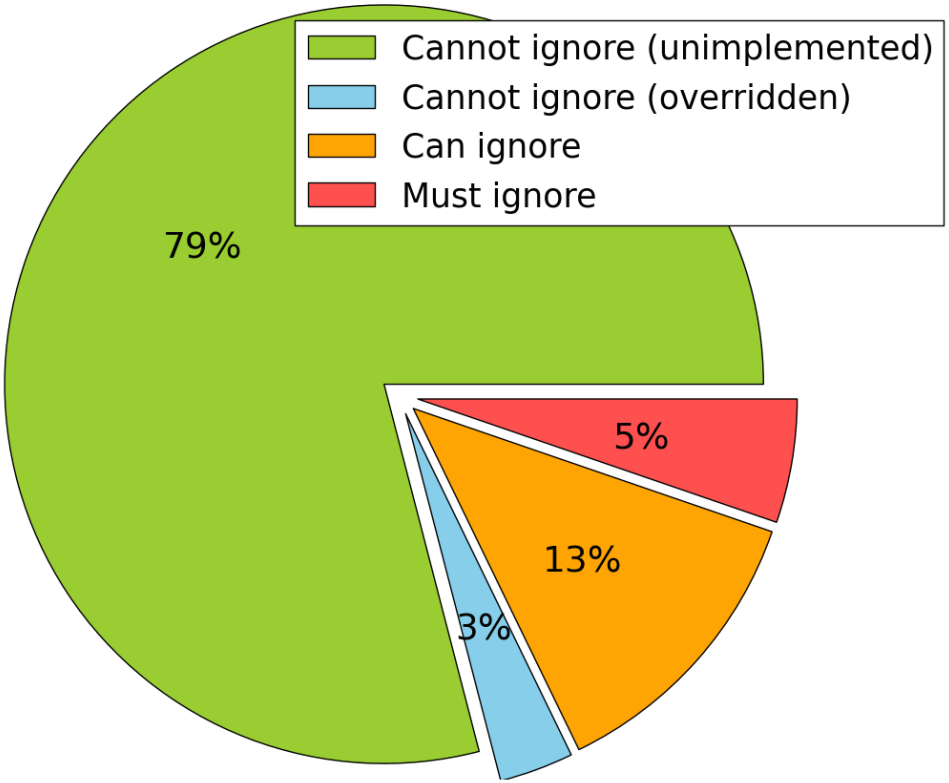
- Severity of exploits in apps that include Bridge Objects and can navigate to content belonging to untrusted origins.

HTTP Severity



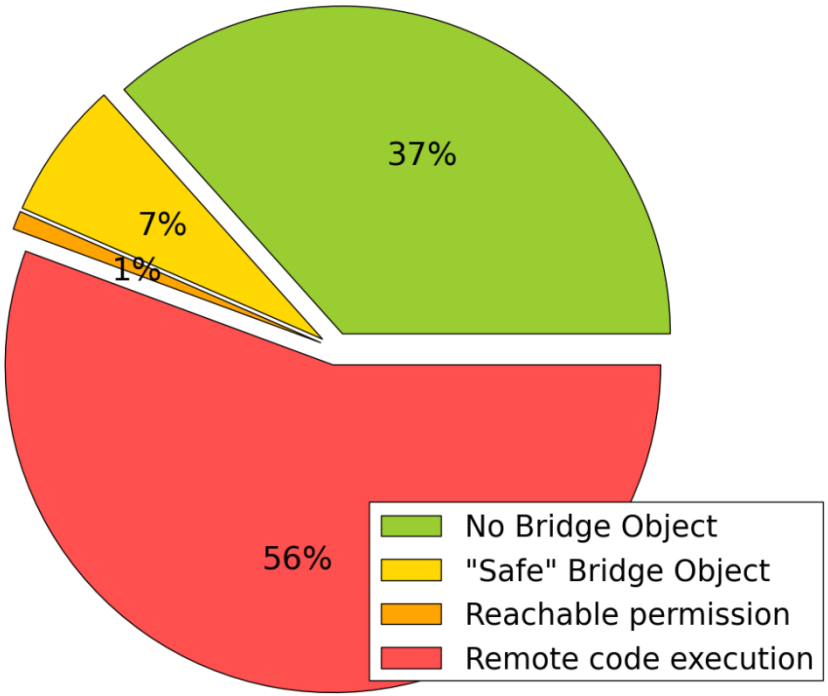
- Severity of Bridge Object exploits in apps that load content over HTTP.

SSL Error Prevalence



- Prevalence of apps that can or must ignore SSL errors based on implementations of onReceivedSslError

SSL Error Severity



- Severity of Bridge Object exploits in apps that must ignore SSL certificate errors.

Lecture Summary

- **General discussion of tools**
  - Fundamental limitations
  - Basic method based on abstract states
- **More details on one specific method**
  - Property checkers from Engler et al., Coverity
  - Sample security-related results
- **Static analysis for Android malware**
  - …

Slides from: S. Bugrahe, A. Chou, I&T Dillig, D. Engler, J. Franklin, A. Aiken,