Automated Tools for System and Application Security

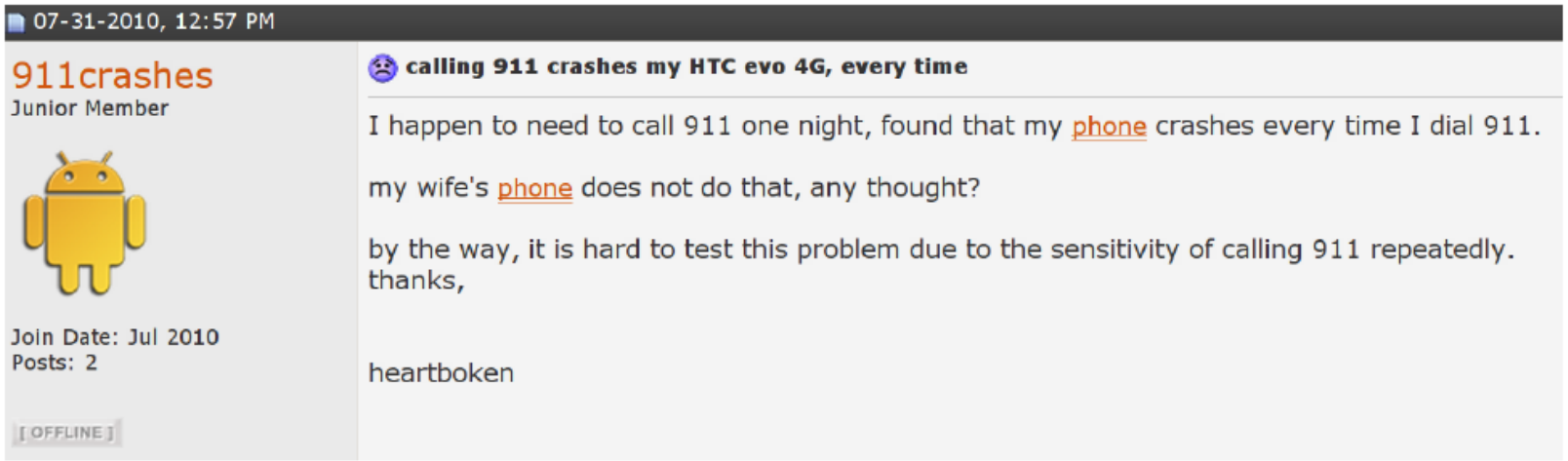
John Mitchell

Outline

- General discussion of code analysis tools
  - Goals and limitations of static, dynamic tools
  - Static analysis based on abstract states
- Security tools for traditional systems programming
  - Property checkers from Engler et al., Coverity
  - Sample security-related results
- Web security analysis
  - Black-box security tools
  - Study based on these tools: security of coding
- Static analysis for Android malware
  - Determining whether app is malicious
  - Using tools for related security studies

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

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

Facebook missed a single security check…

[PopPhoto.com Feb 10]

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.

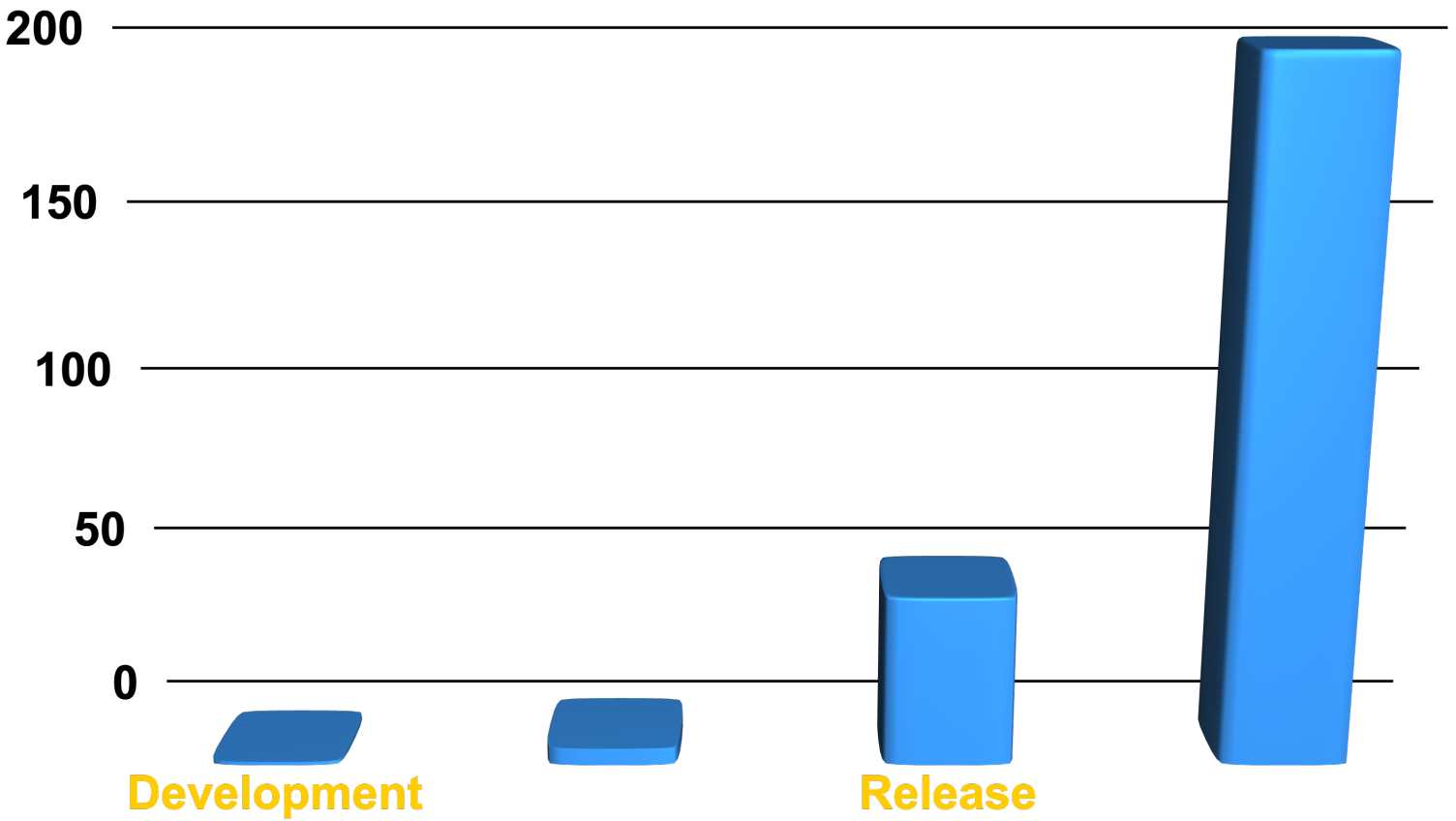
View food and drink apps in the App Store ›

How can you tell whether software you

- Develop
- Buy

is safe to install and run?

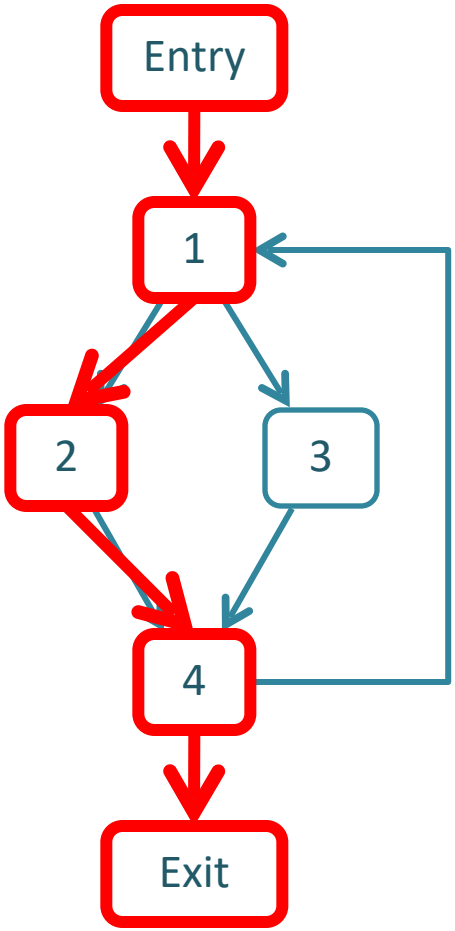
Cost of Fixing a Defect



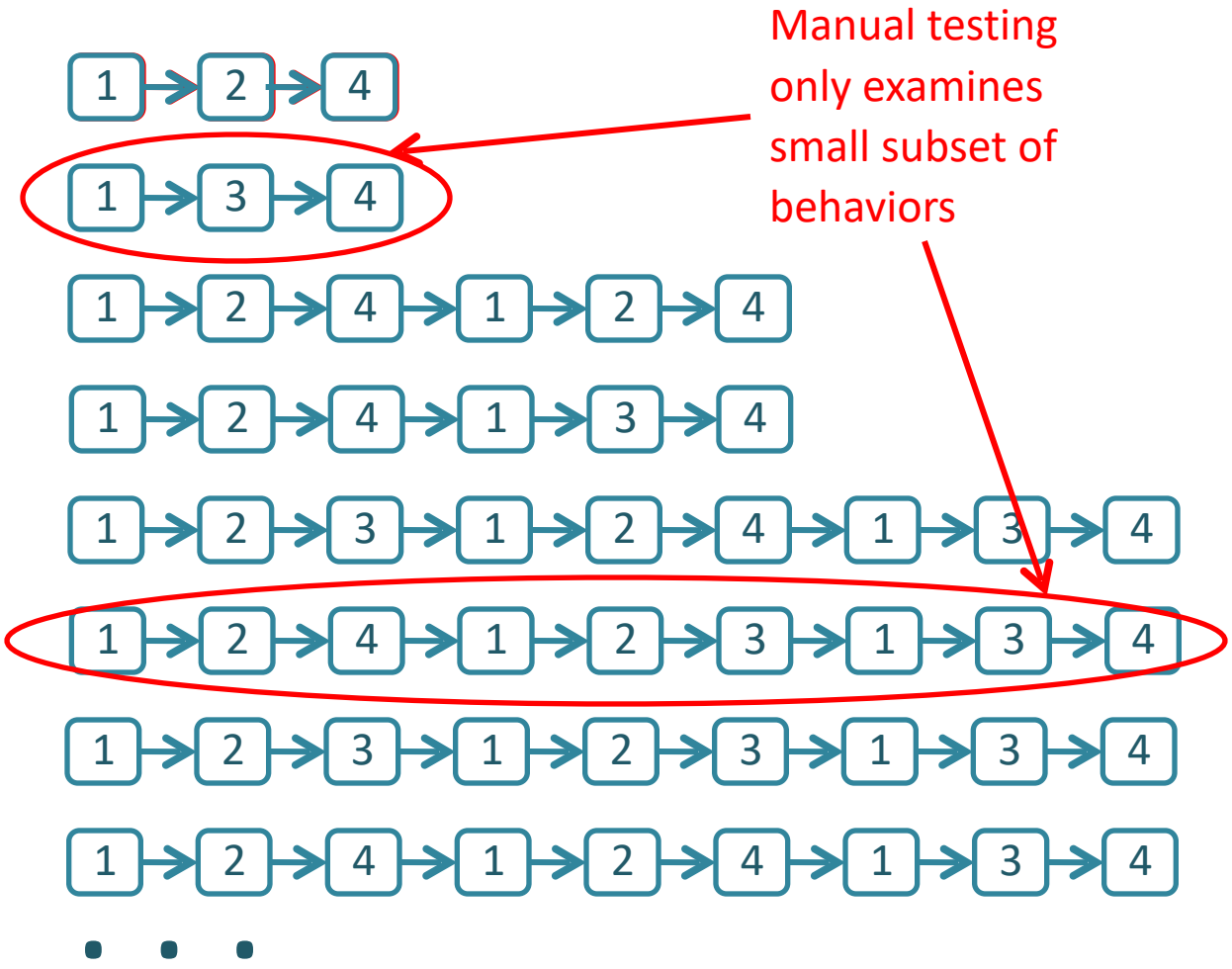
Credit: Andy Chou, Coverity

Cost of security or data privacy vulnerability?

Tools to help you

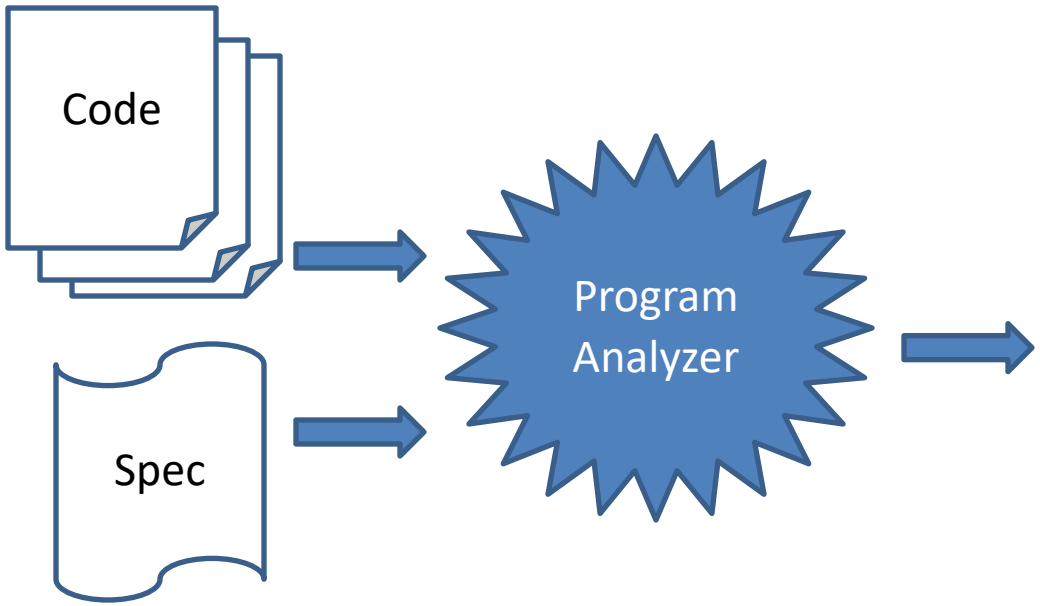


Software



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

Two options

- Static analysis
  - Automated methods to find errors or check their absence
    - Consider all possible inputs (in summary form)
    - Find bugs and vulnerabilities
    - Can prove absence of bugs, in some cases
- Dynamic analysis
  - Run instrumented code to find problems
    - Need to choose sample test input
    - Can find vulnerabilities but cannot prove their absence

Static Analysis

- Long research history
- Decades of commercial products
  - FindBugs, Fortify, Coverity, MS tools, …

Dynamic analysis

- 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

Summary

- Program analyzers
  - Find problems in code before it is shipped to customers or before you install and run it
- Static analysis
  - Analyze code to determine behavior on all inputs
- Dynamic analysis
  - Choose some sample inputs and run code to see what happens

Outline

- General discussion of code analysis tools
  - Goals and limitations of static, dynamic tools
  - Static analysis based on abstract states
- Security tools for traditional systems programming
  - Property checkers from Engler et al., Coverity
  - Sample security-related results
- Web security analysis
  - Black-box security tools
  - Study based on these tools: security of coding
- Static analysis for Android malware
  - Determining whether app is malicious
  - Using tools for other security studies

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

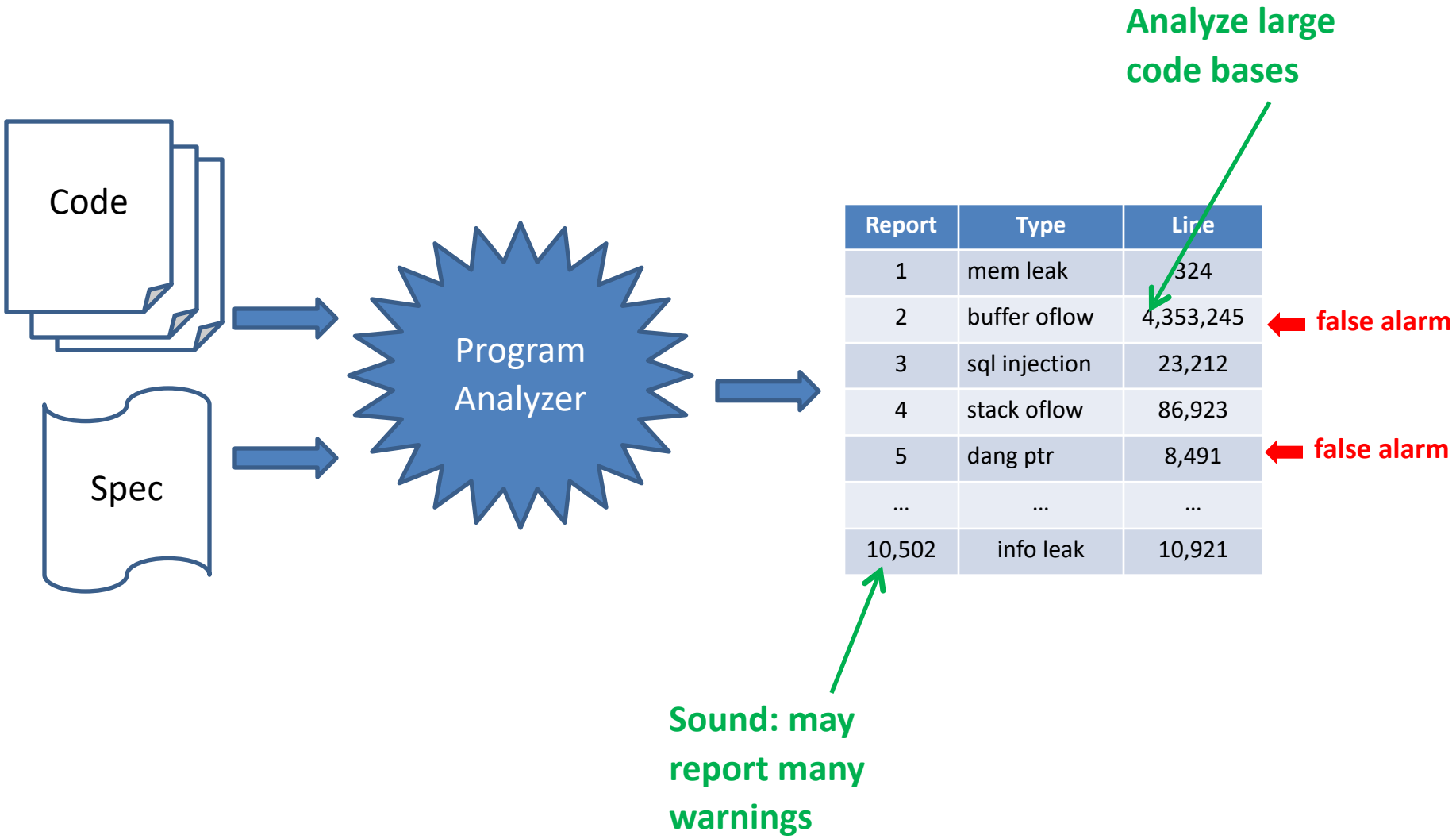
Soundness, Completeness

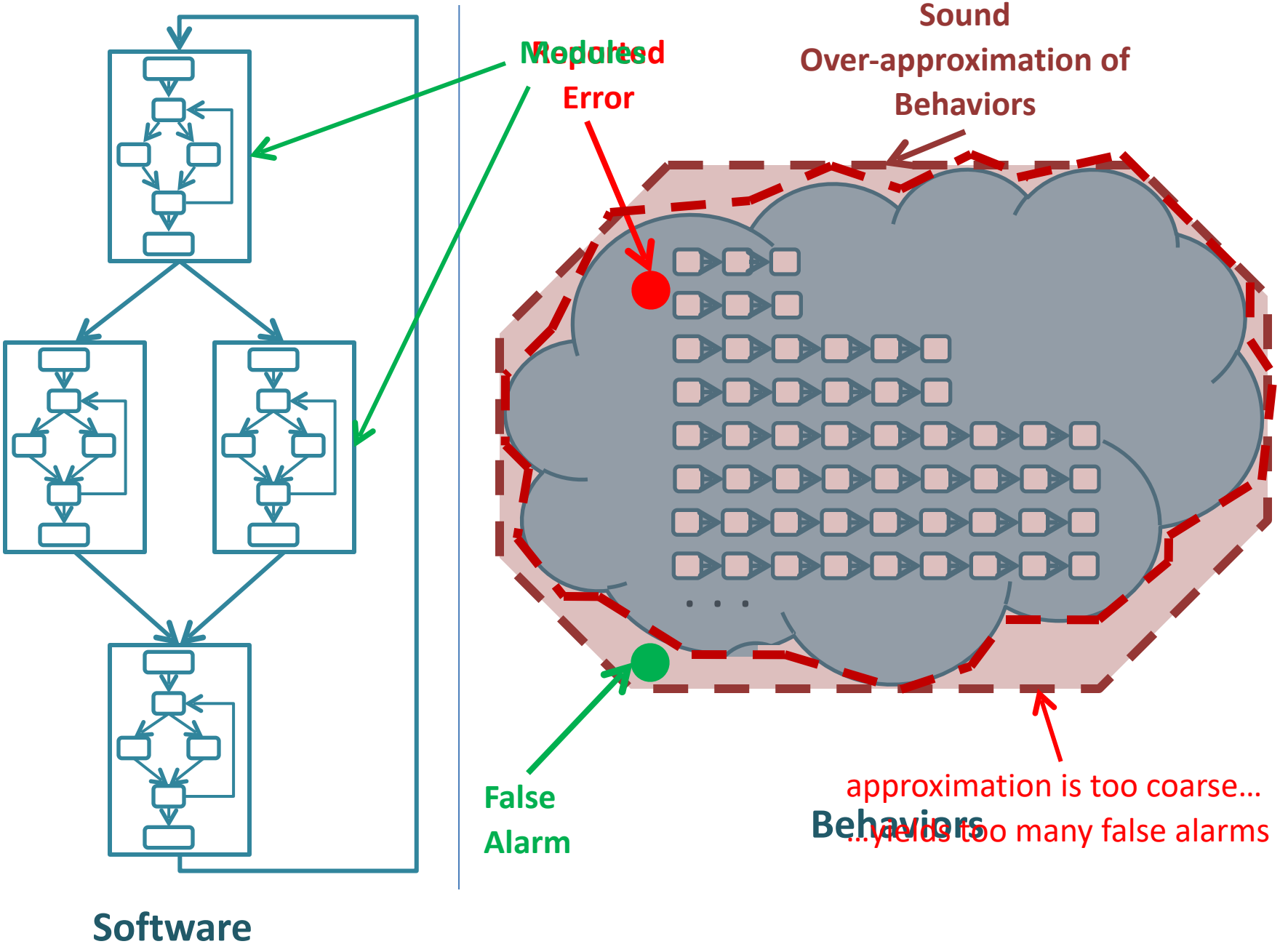
Property	Definition
Soundness	"Sound for reporting correctness" Analysis says no bugs → No bugs or equivalently There is a bug → Analysis finds a bug
Completeness	"Complete for reporting correctness" No bugs → Analysis says no bugs

Recall: $A \rightarrow B$ is equivalent to $(\neg B) \rightarrow (\neg A)$

	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**

Sound Program Analyzer

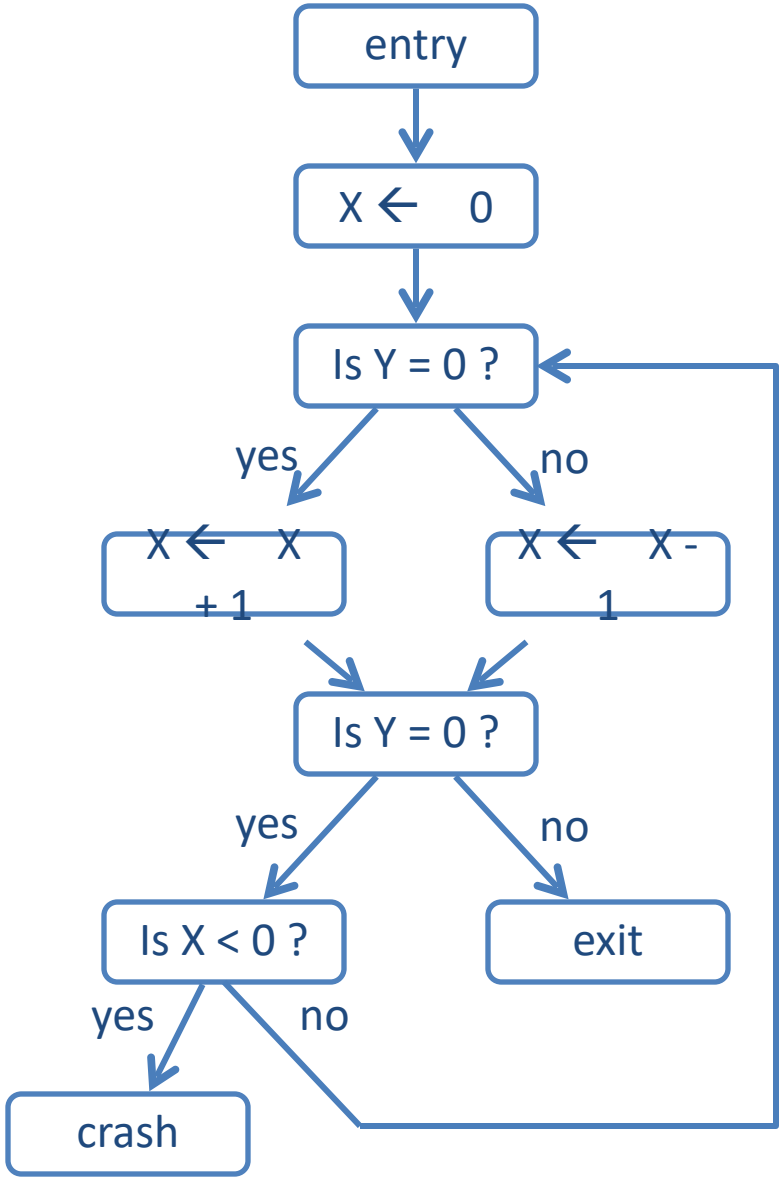



Sound
Over-approximation of
Behaviors
Modeled
Error
False
Alarm
approximation is too coarse…
…yields too many false alarms
Behaviors
Software

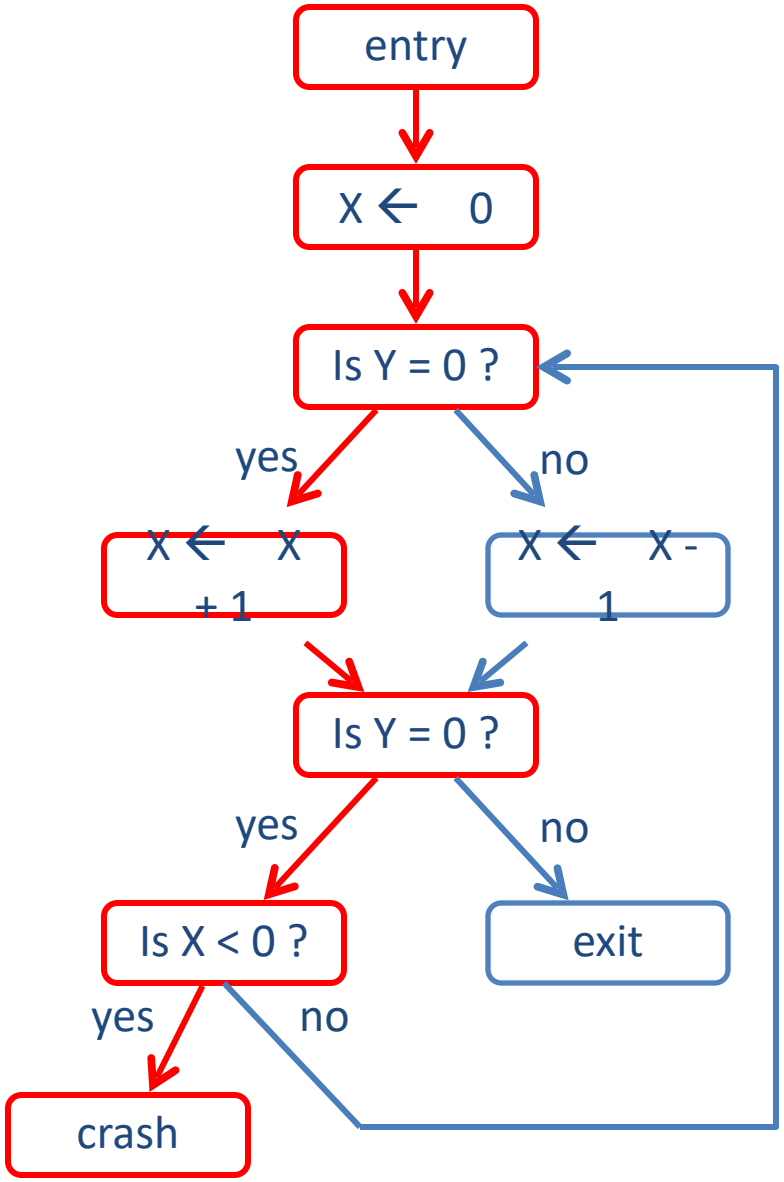
EXAMPLE

Program execution based on abstract states

Does this program ever crash?



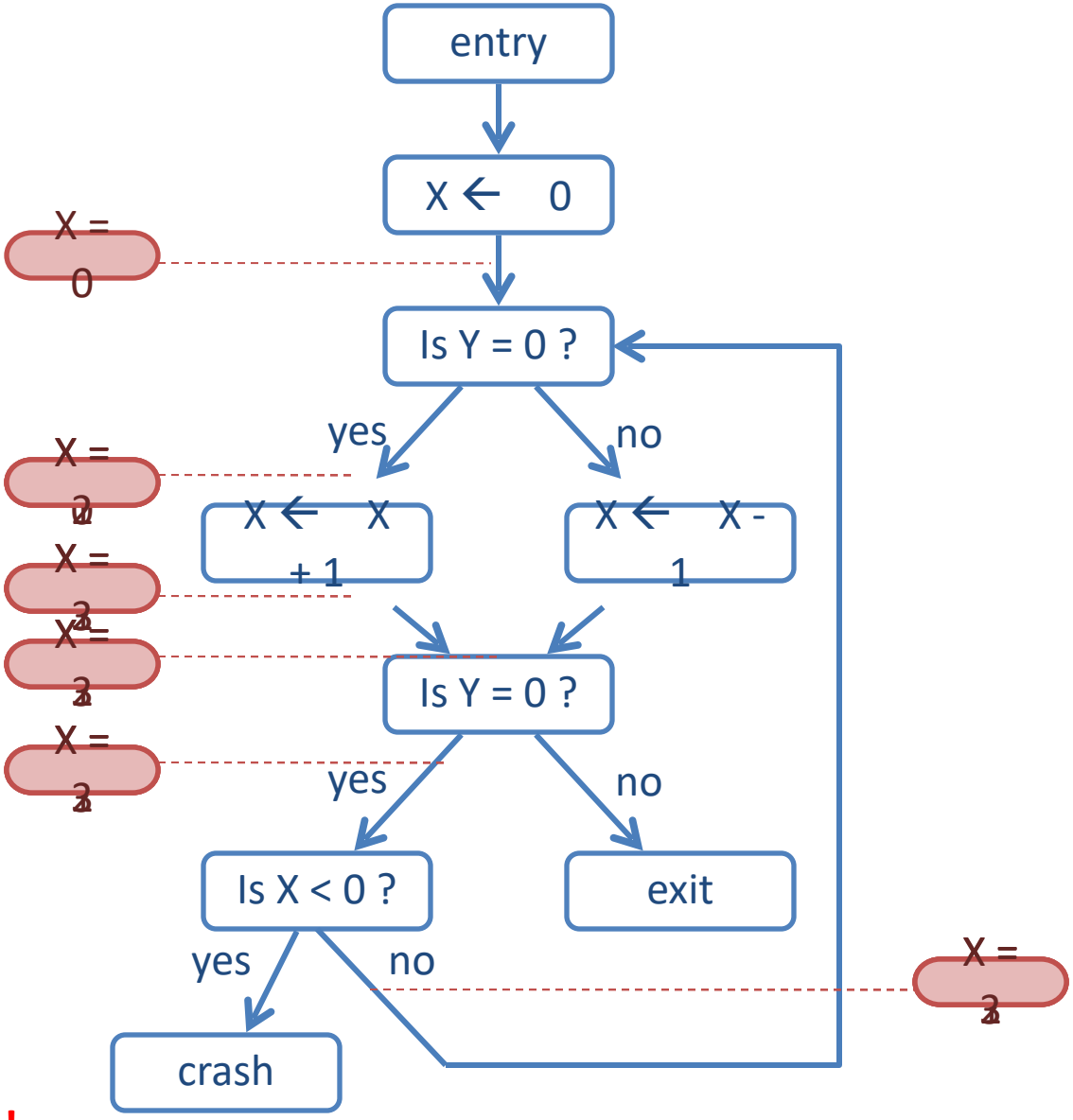
Does this program ever crash?



infeasible path!

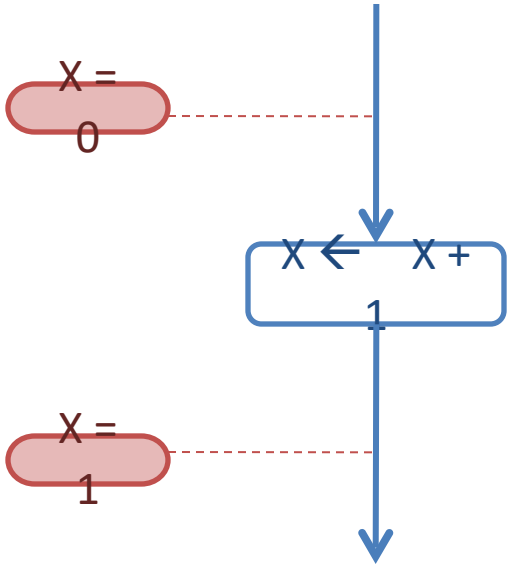
… program will never crash

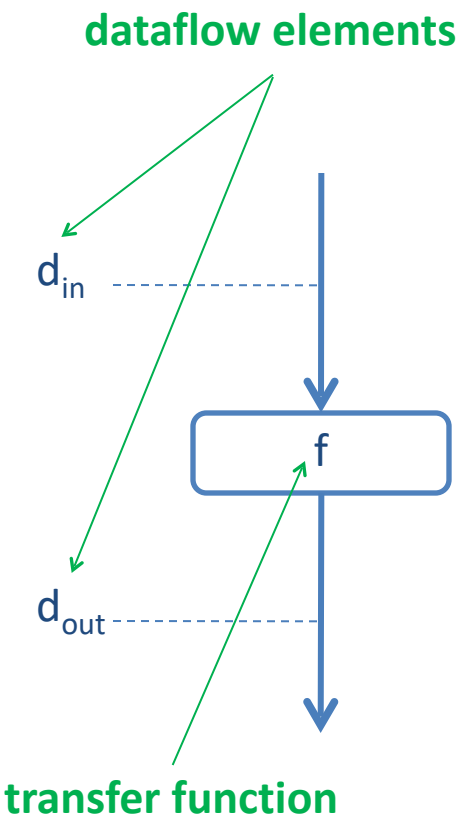
Try analyzing without approximating…



non-termination!

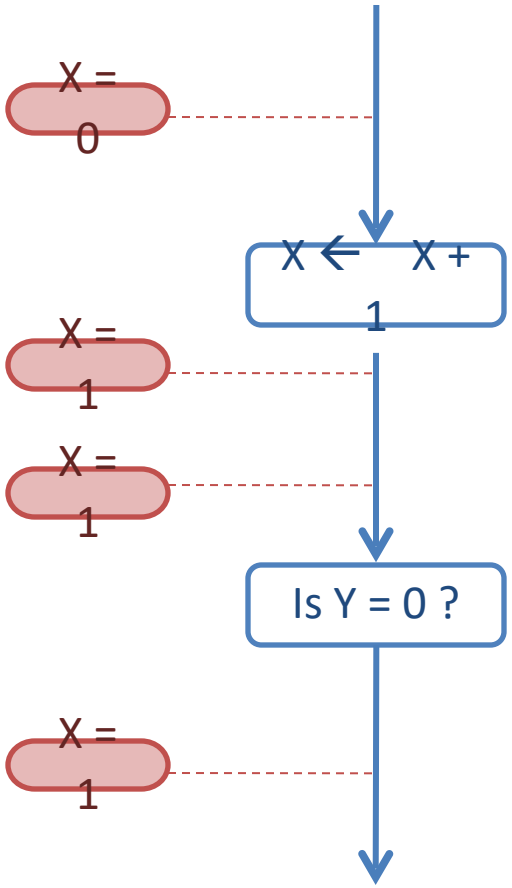
… therefore, need to approximate

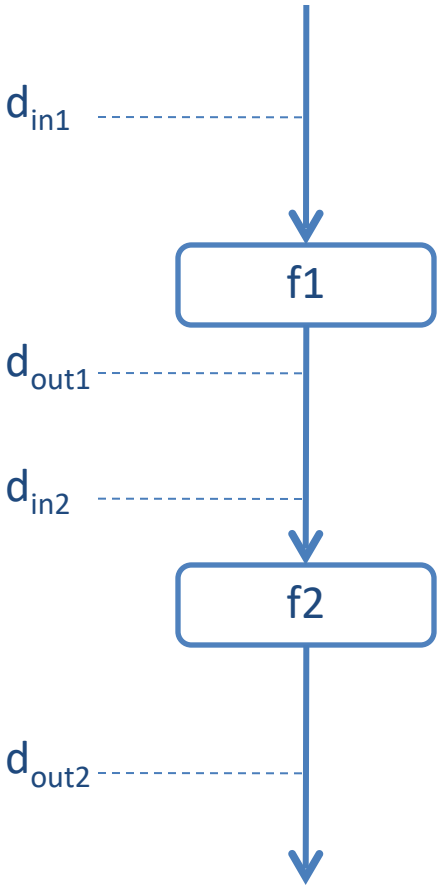




$$d_{out} = f(d_{in})$$

dataflow equation

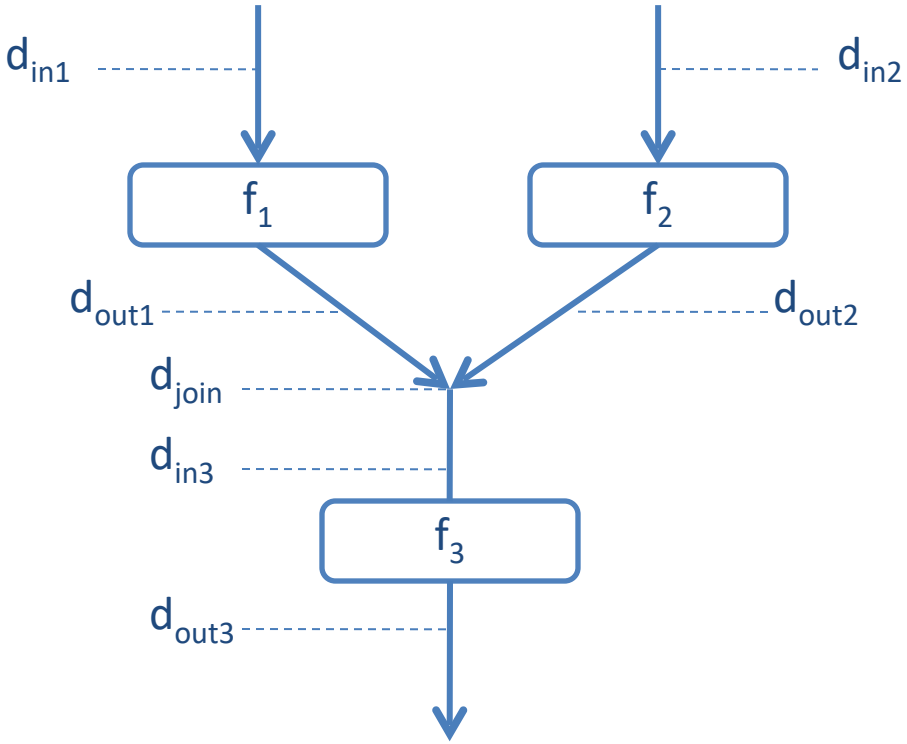




$$d_{out1} = f_1(d_{in1})$$

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

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



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

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

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

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

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

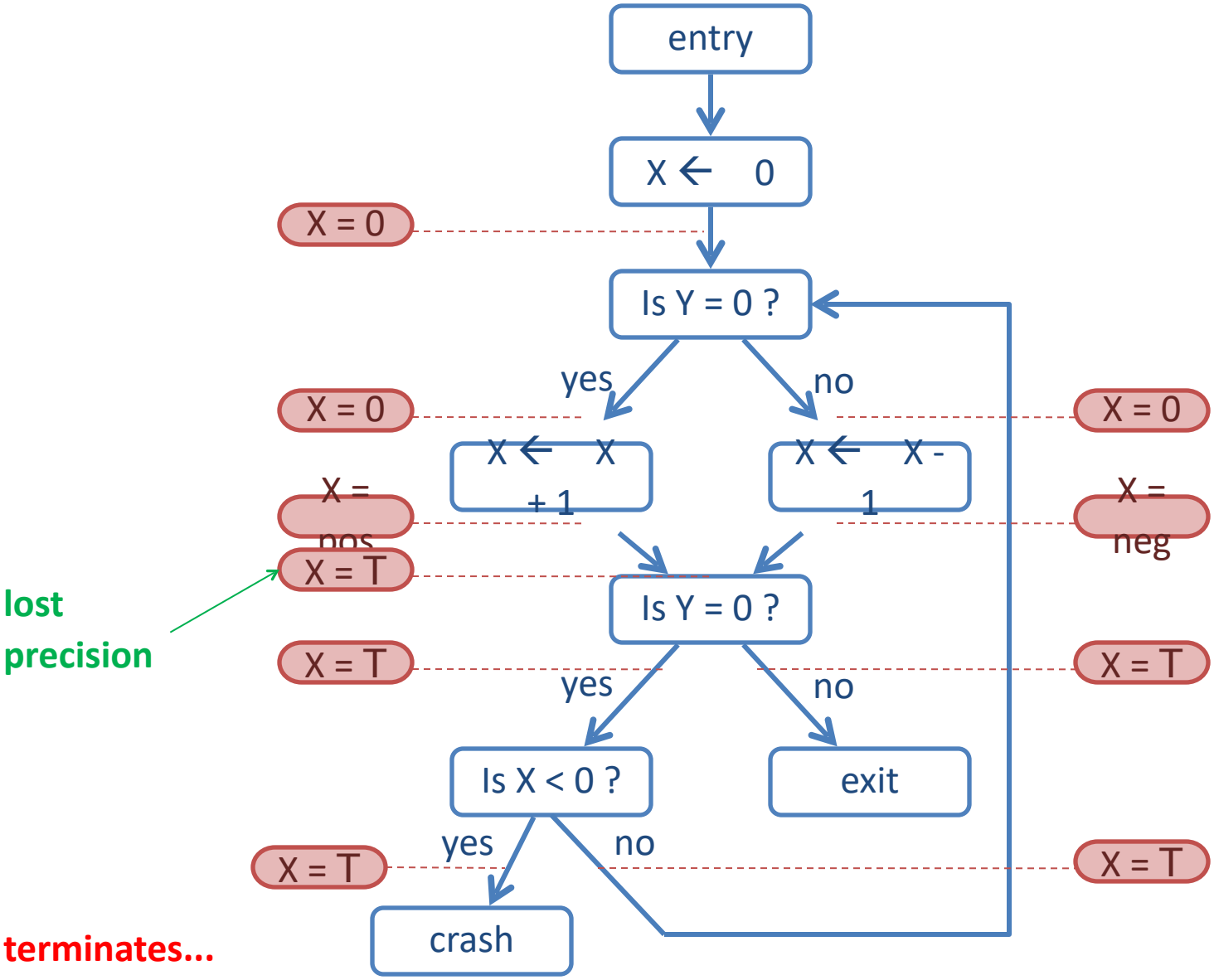
least upper bound operator

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…

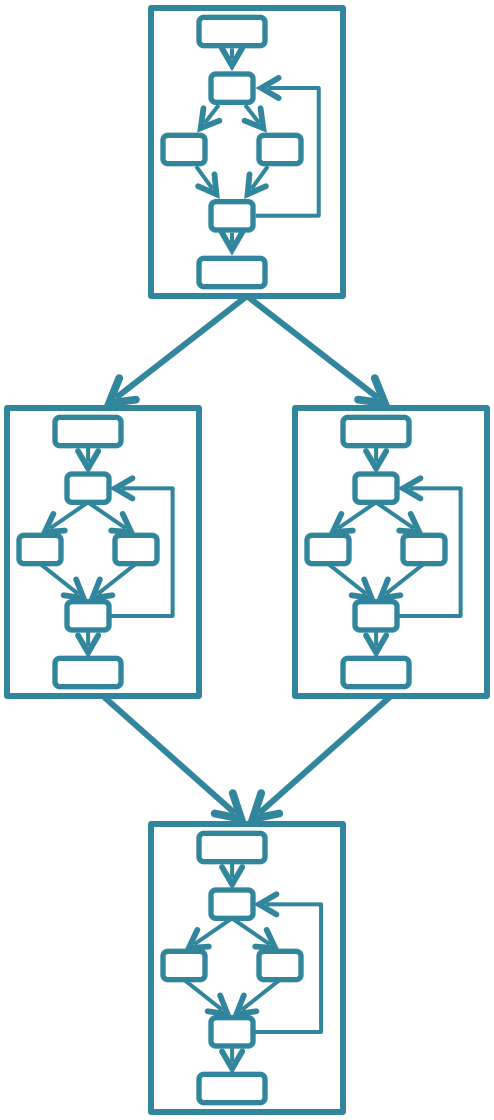


lost precision

terminates...

… but reports false alarm

… therefore, need more precision



Software

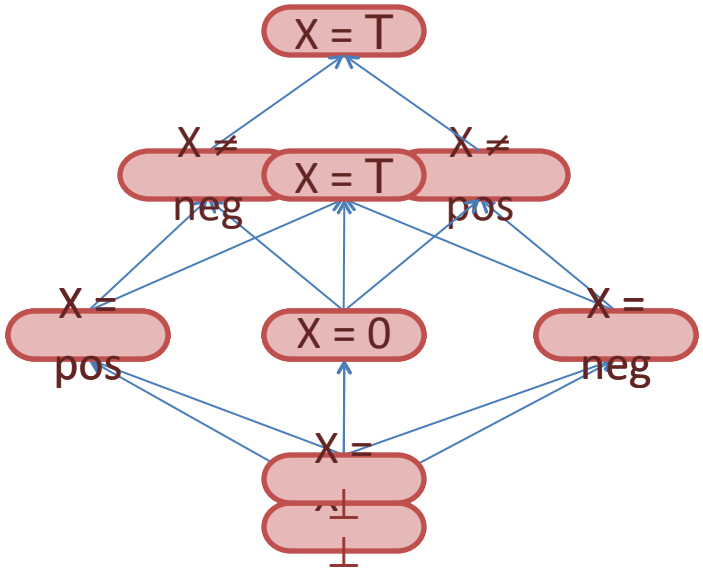
Reported Error

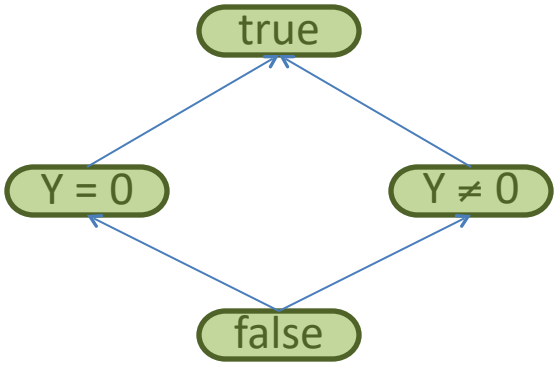
Sound Over-approximation of Behaviors

False Alarm

Behaviors

approximation is too coarse…
…yields too many false alarms

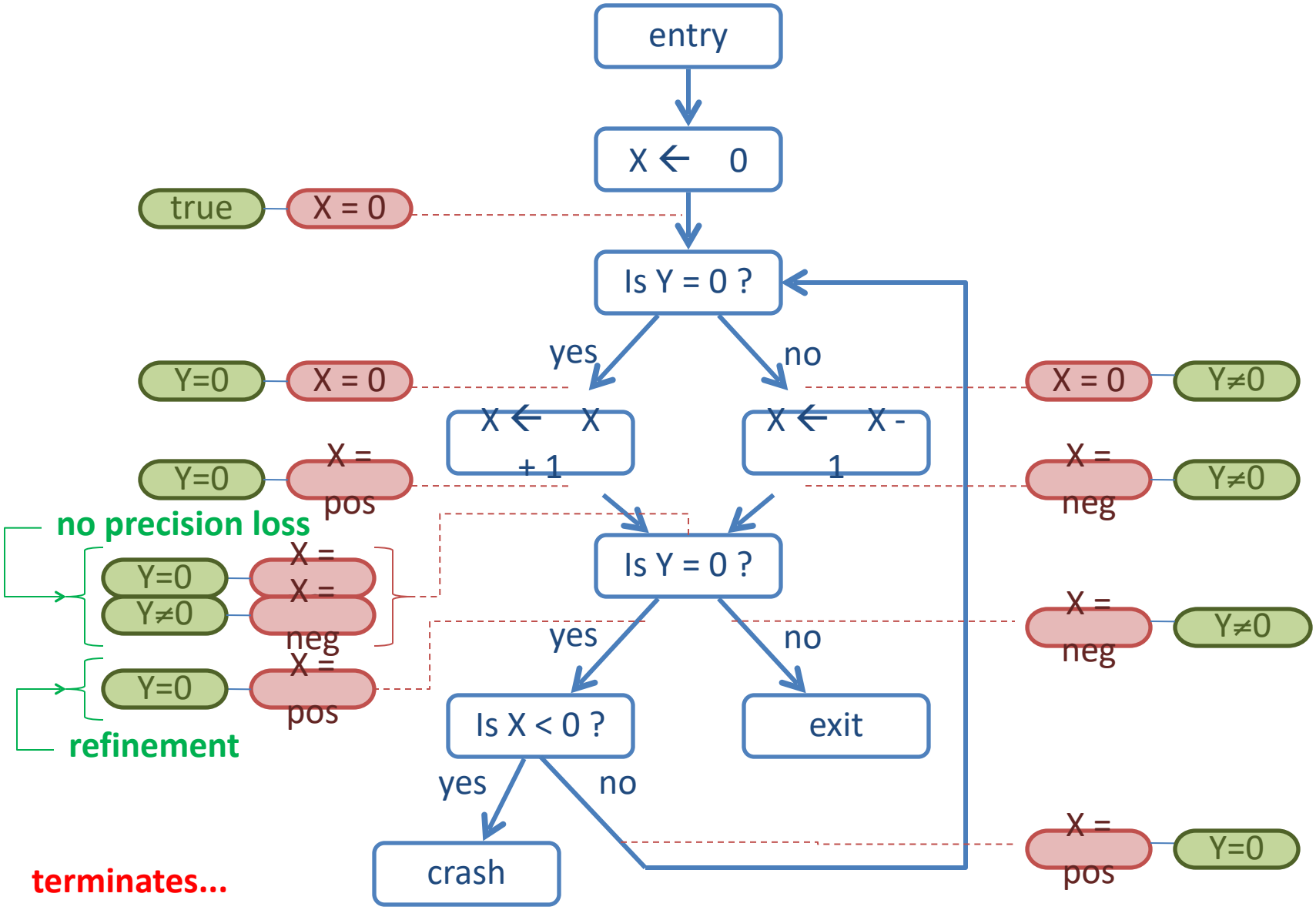




refinement lattice

Boolean formula lattice

Try analyzing with “path-sensitive signs” approximation…



terminates...

… no false alarm

… soundly proved never crashes

Summary of *sound* analysis

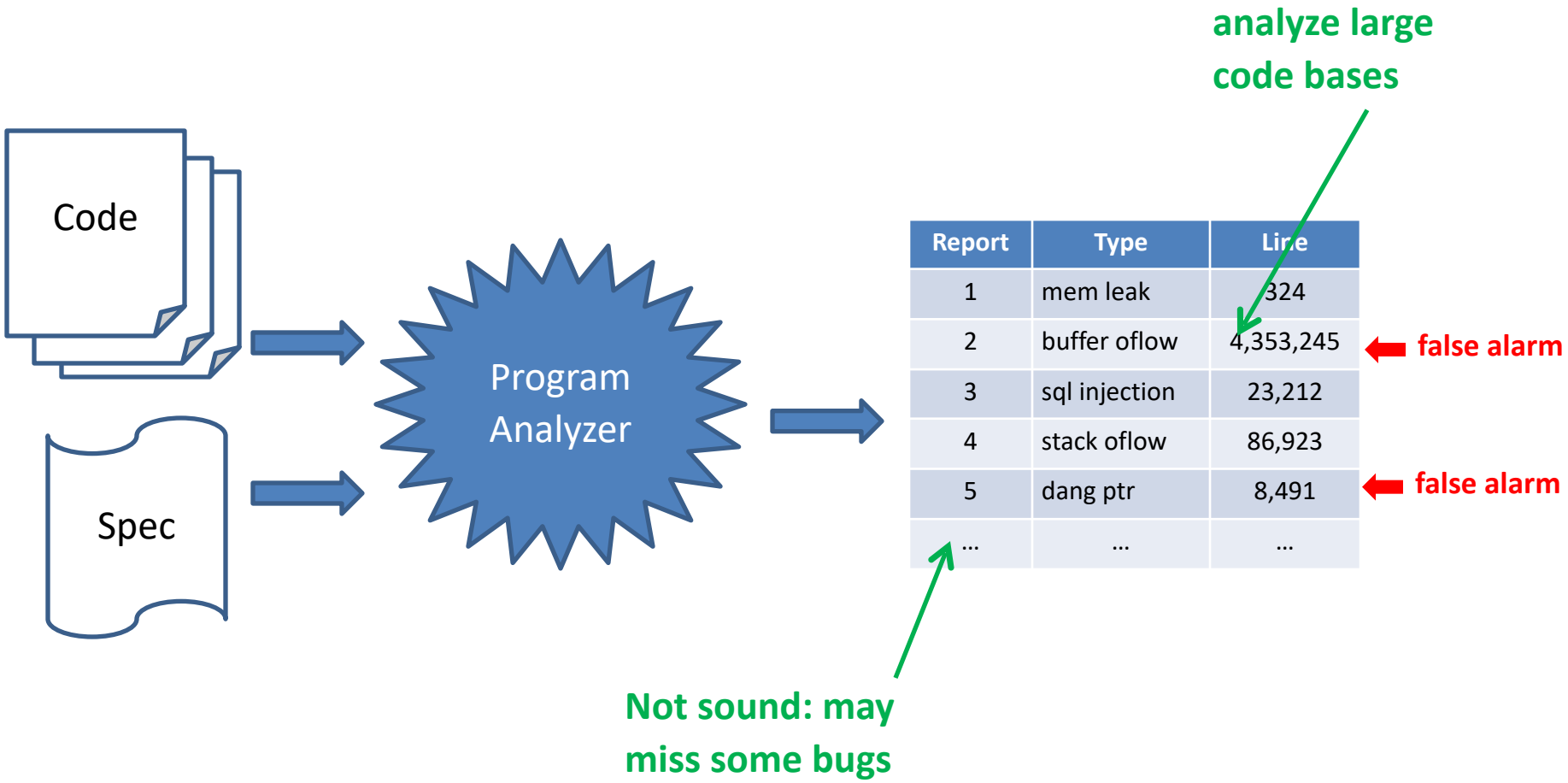
- Sound vs Complete
  - Cannot be sound and complete
  - Sound: can guarantee absence of bugs
- Sound analysis based on abstract states
  - Symbolically execute code using a description of all possible states at this program point
  - Better description: more precise, less efficient
- In practice
  - Use basic approach, possibly without soundness
    - E.g., do not run loops to termination
  - But keep the example in mind as good illustration

Outline

- General discussion of code analysis tools
  - Goals and limitations of static, dynamic tools
  - Static analysis based on abstract states
- Security tools for traditional systems programming
  - Property checkers from Engler et al., Coverity
  - Sample security-related results
- Web security analysis
  - Black-box security tools
  - Study based on these tools: security of coding
- Static analysis for Android malware
  - Determining whether app is malicious
  - Using tools for other security studies

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

Unsound Program Analyzer



Bugs to Detect

Some examples

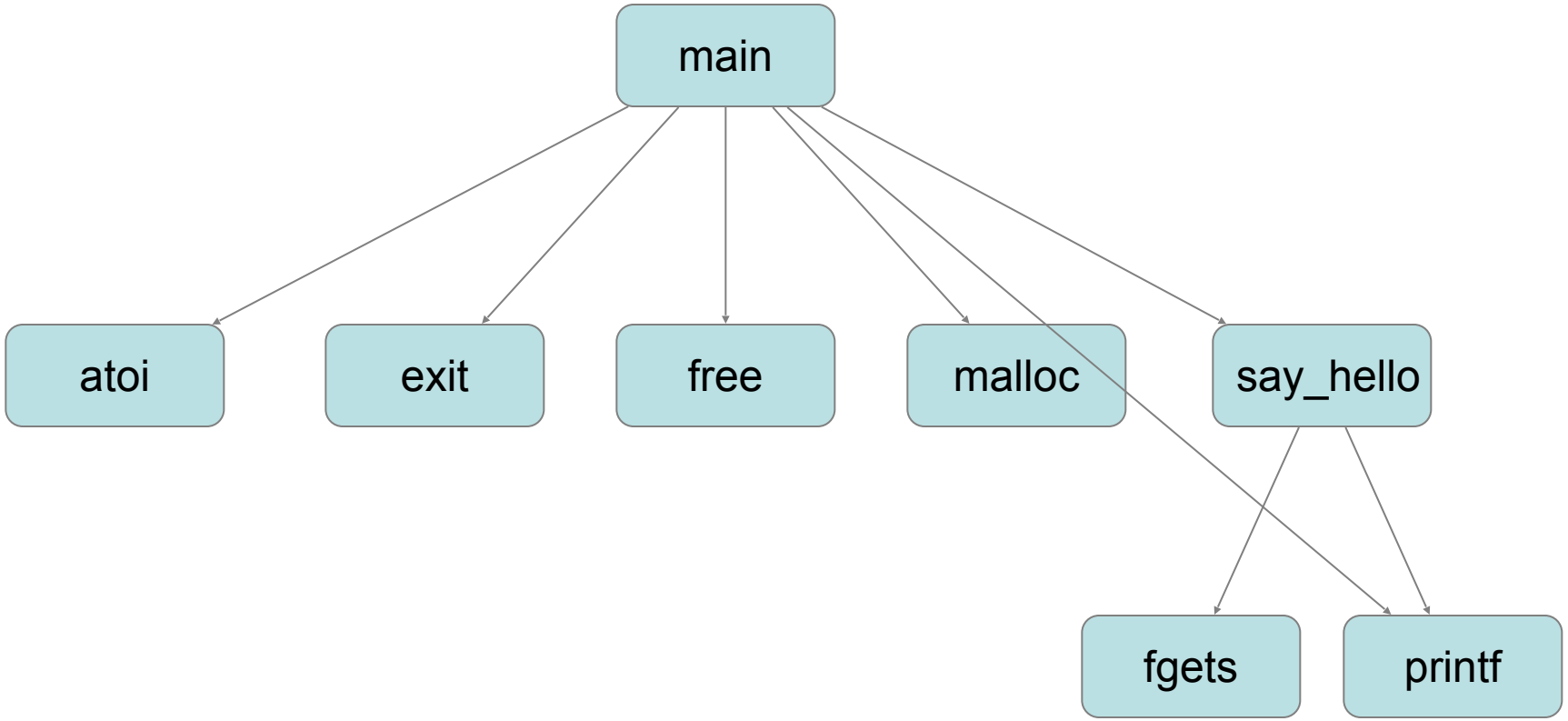
- Crash Causing Defects
- Null pointer dereference
- Use after free
- Double free
- Array indexing errors
- Mismatched array new/delete
- Potential stack overrun
- Potential heap overrun
- Return pointers to local variables
- Logically inconsistent code
- Uninitialized variables
- Invalid use of negative 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

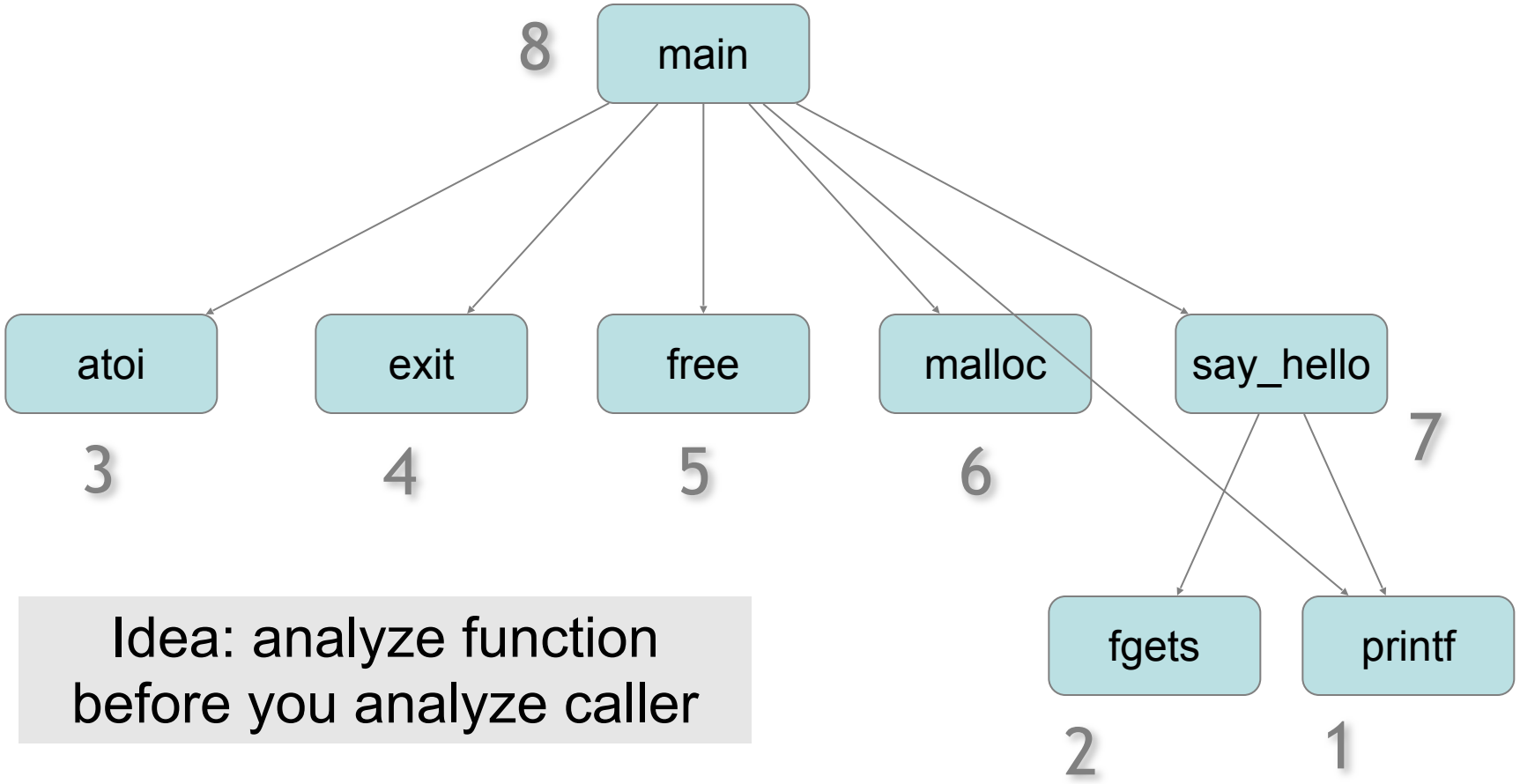
Example code with function def, calls

```
#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

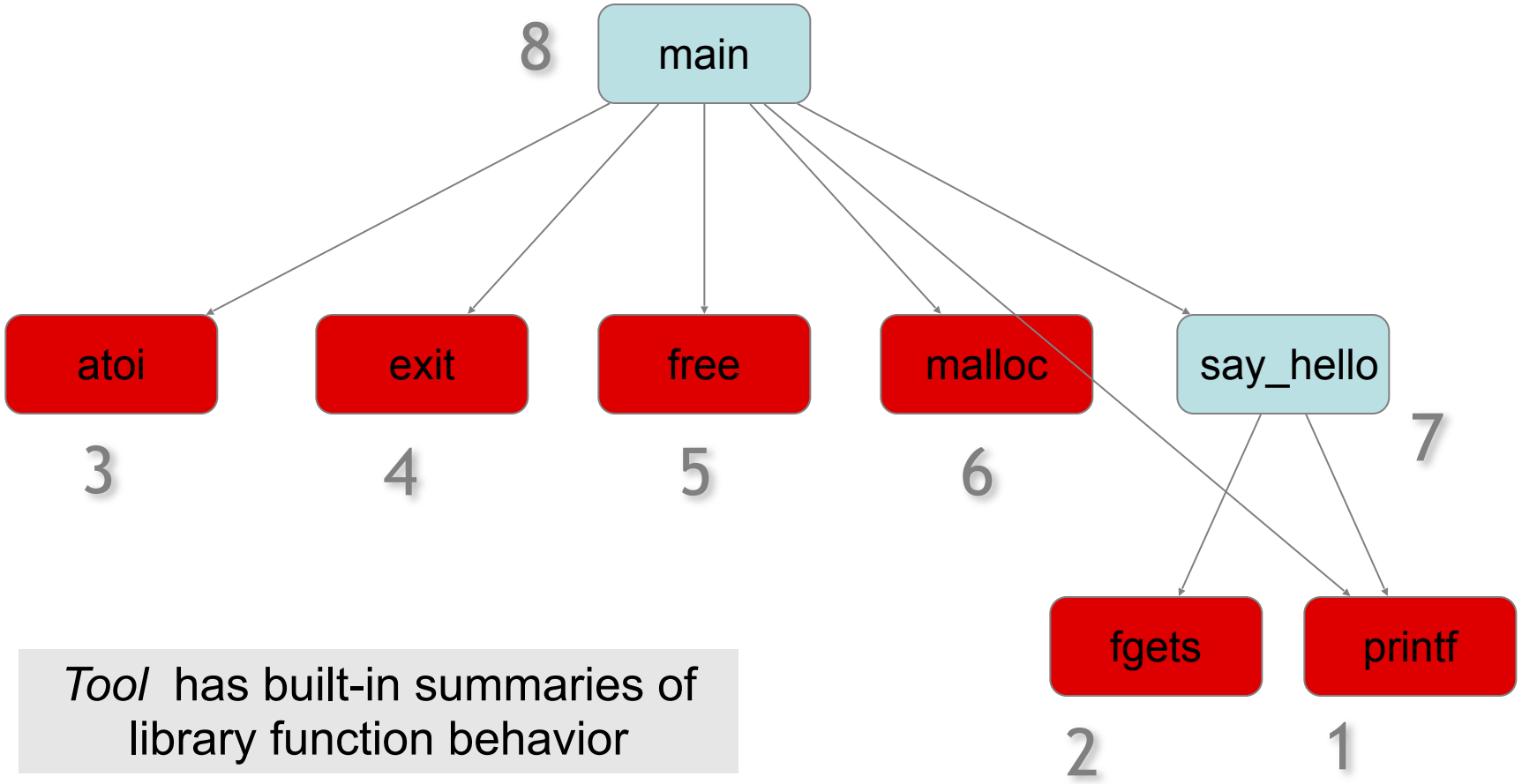


Reverse Topological Sort



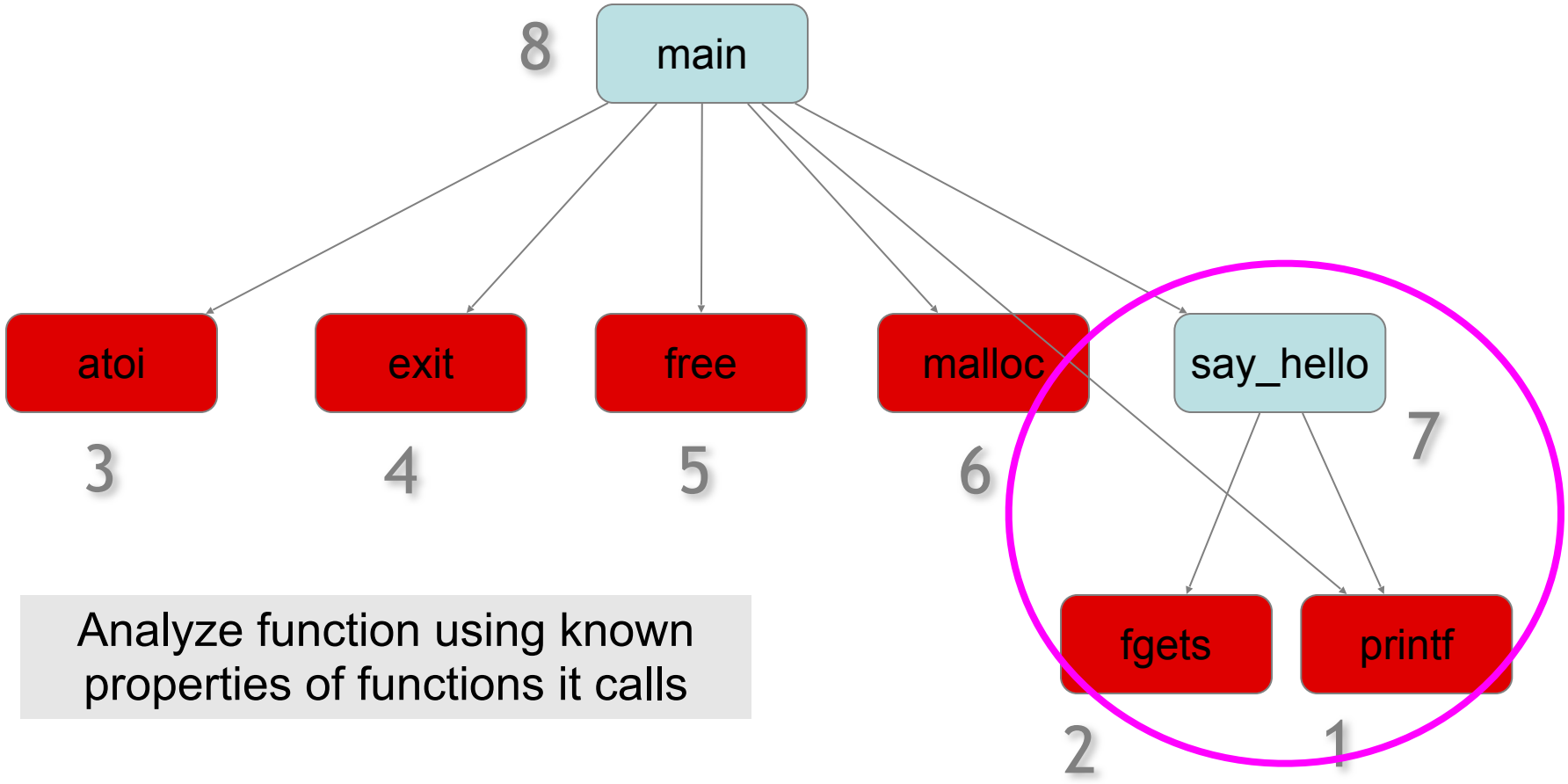
Idea: analyze function before you analyze caller

Apply Library Models



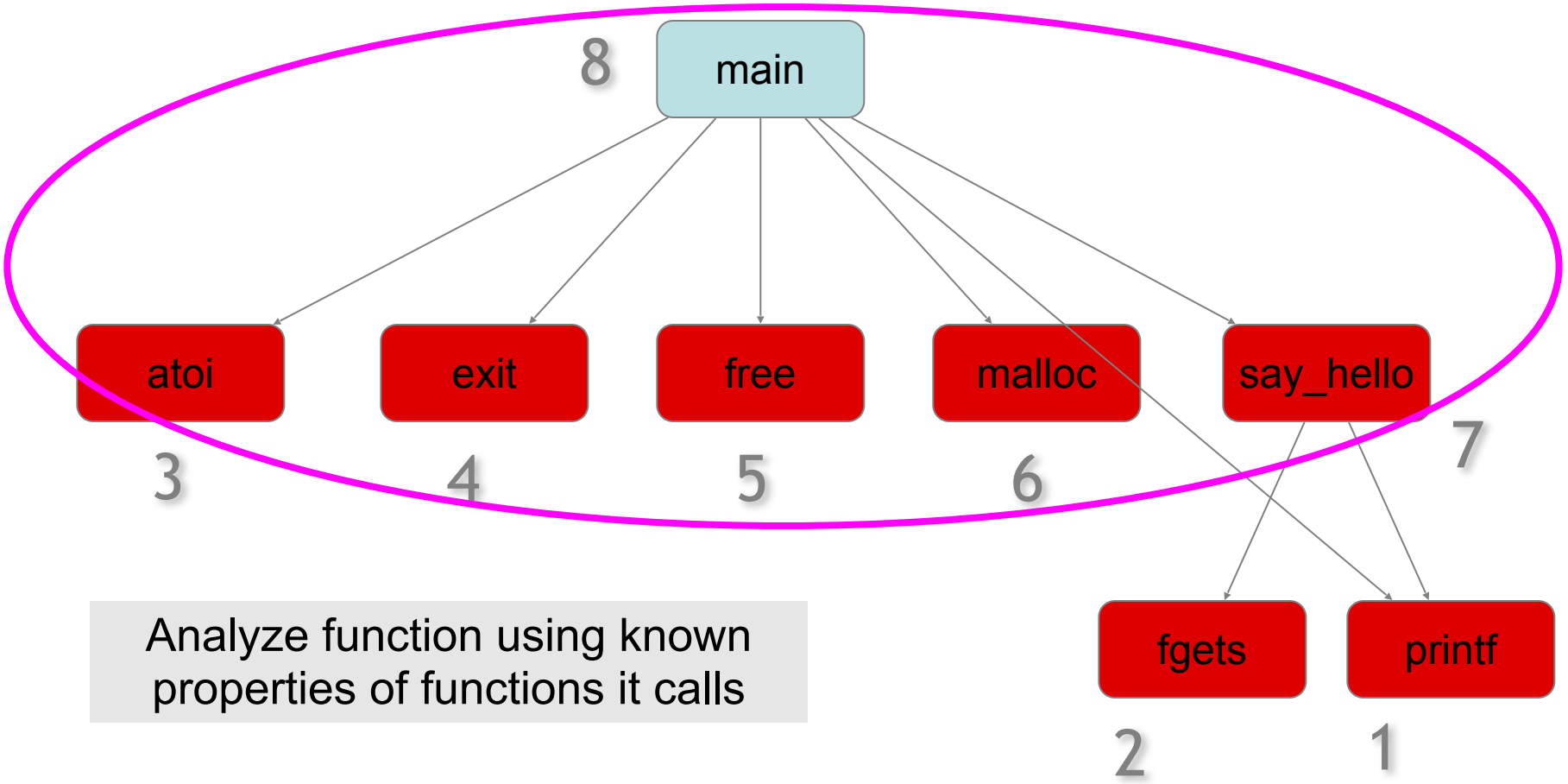
Tool has built-in summaries of library function behavior

Bottom Up Analysis



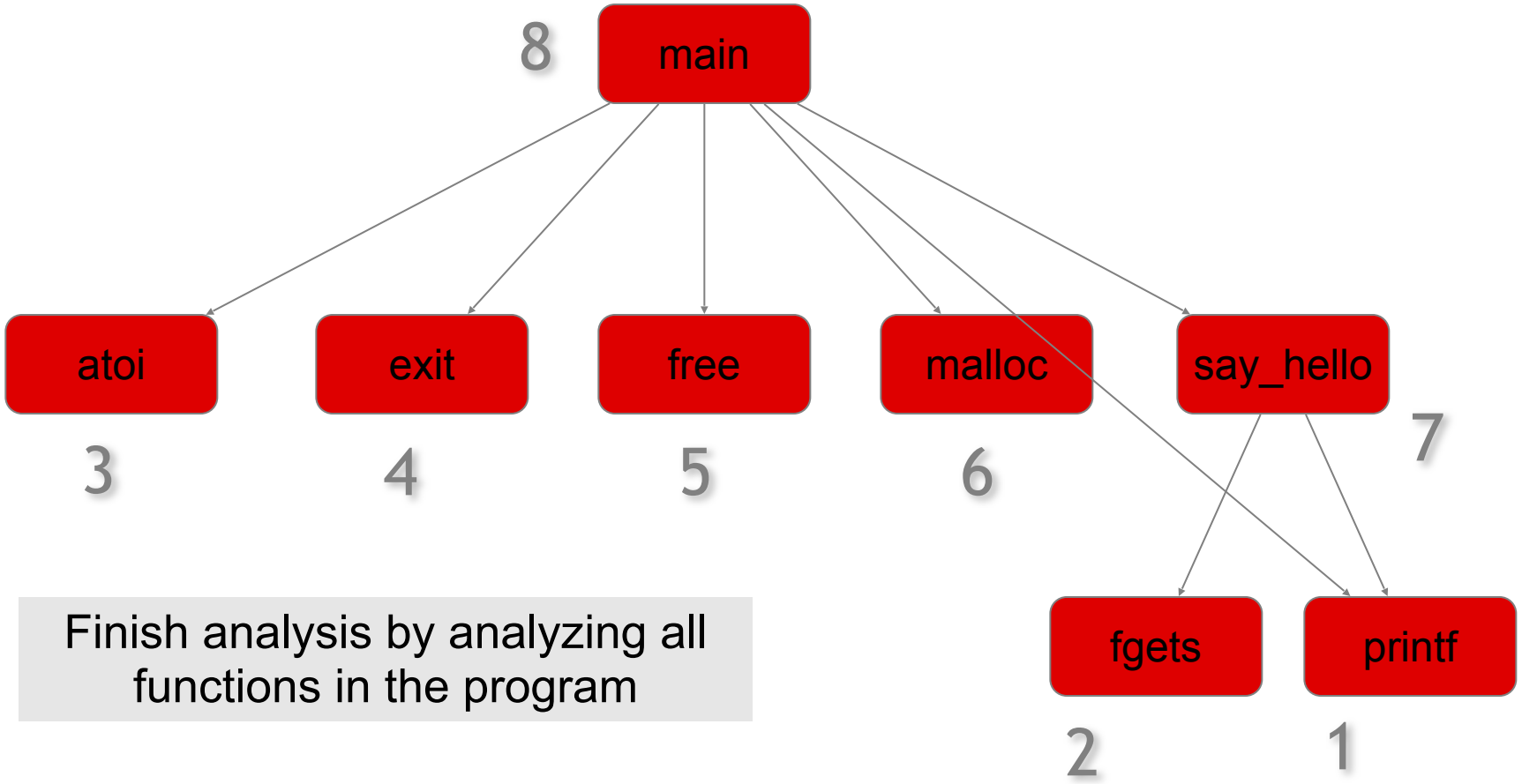
Analyze function using known properties of functions it calls

Bottom Up Analysis



Analyze function using known properties of functions it calls

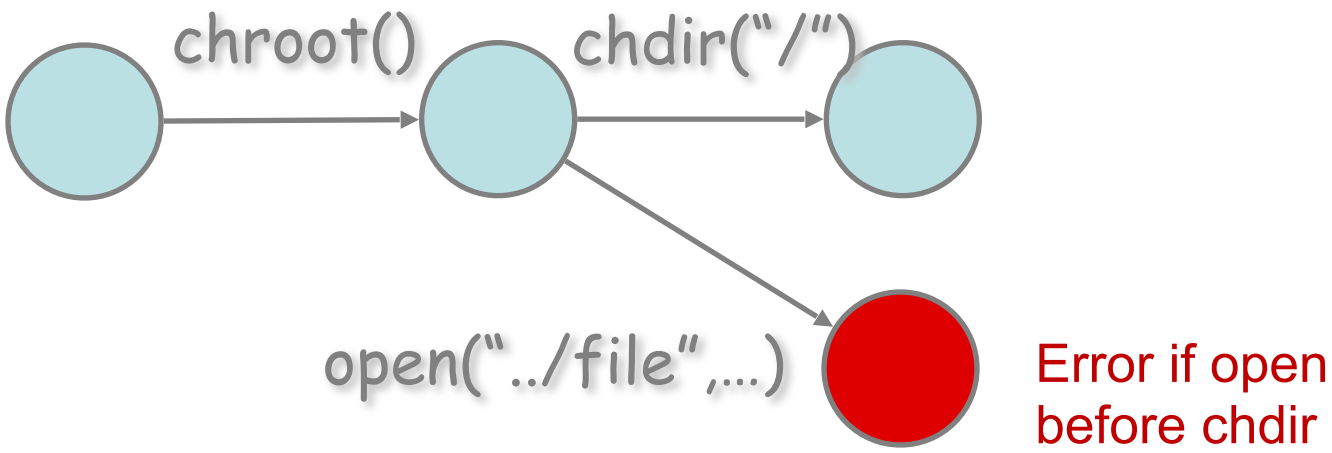
Bottom Up Analysis



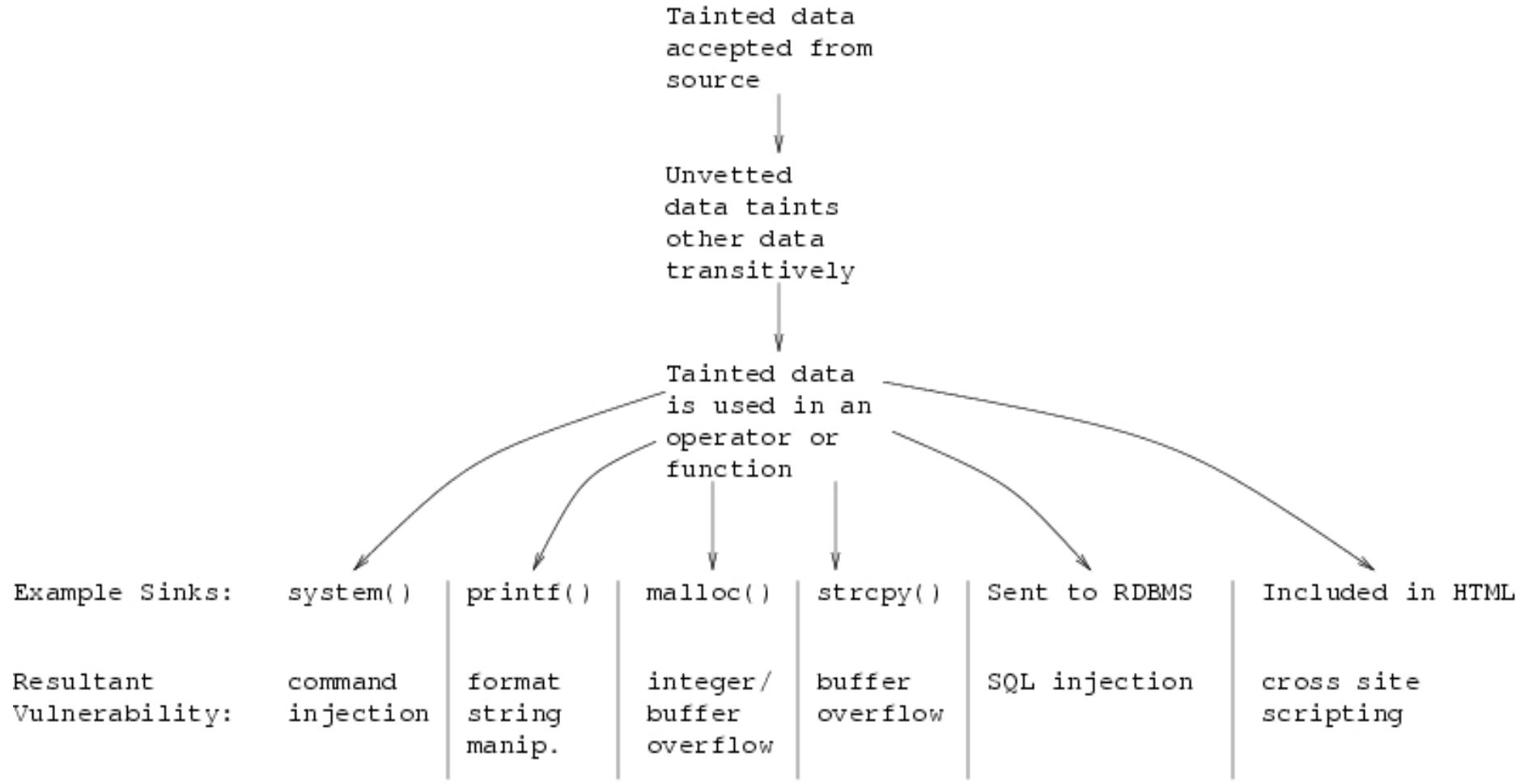
Finish analysis by analyzing all functions in the program

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



Tainting checkers

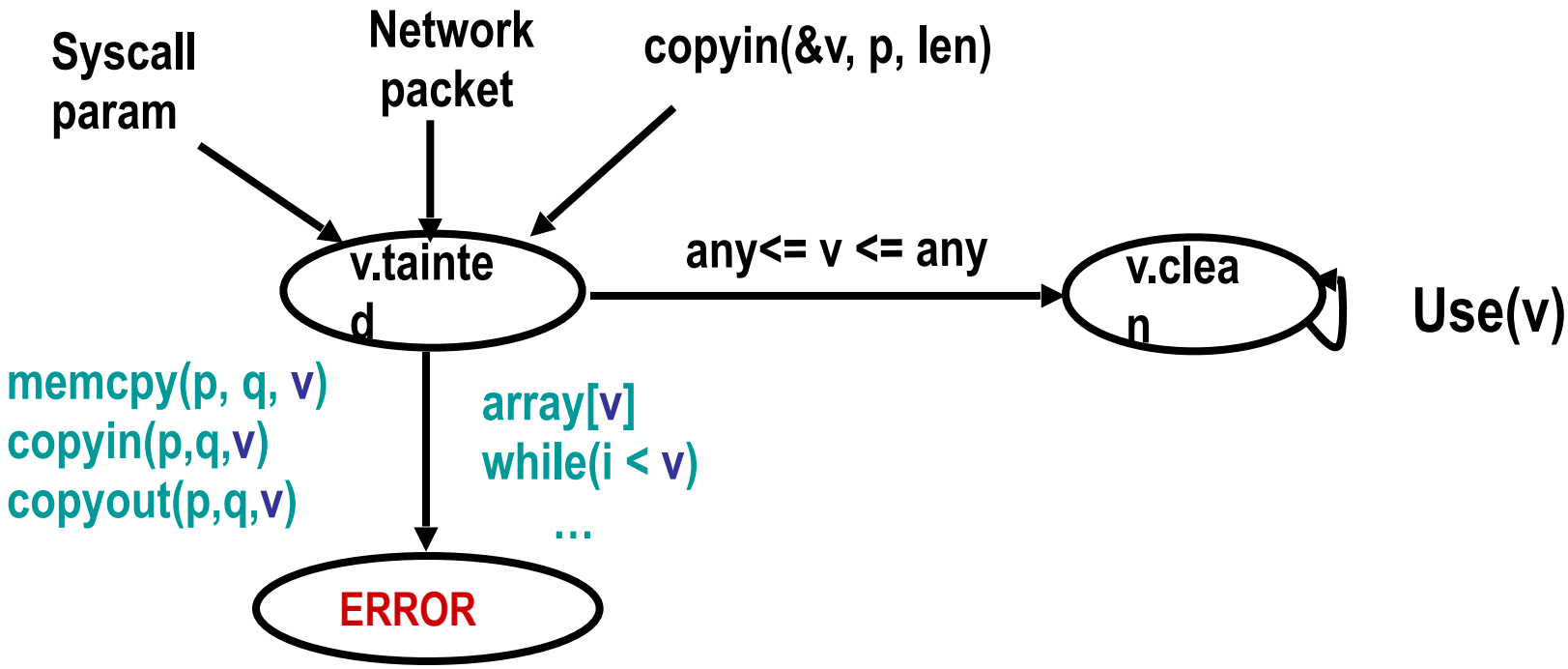


Application to Security Bugs

- **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);
```

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;
```

```
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;
```

1/4 deref

Results for BSD and Linux

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

Violation	Linux		BSD	
	Bug	Fixed	Bug	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 code analysis tools
  - Goals and limitations of static, dynamic tools
  - Static analysis based on abstract states
- Security tools for traditional systems programming
  - Property checkers from Engler et al., Coverity
  - Sample security-related results
- Web security analysis
  - Black-box security tools
  - Study based on these tools: security of coding
- Static analysis for Android malware
  - Determining whether app is malicious
  - Using tools for other security studies

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

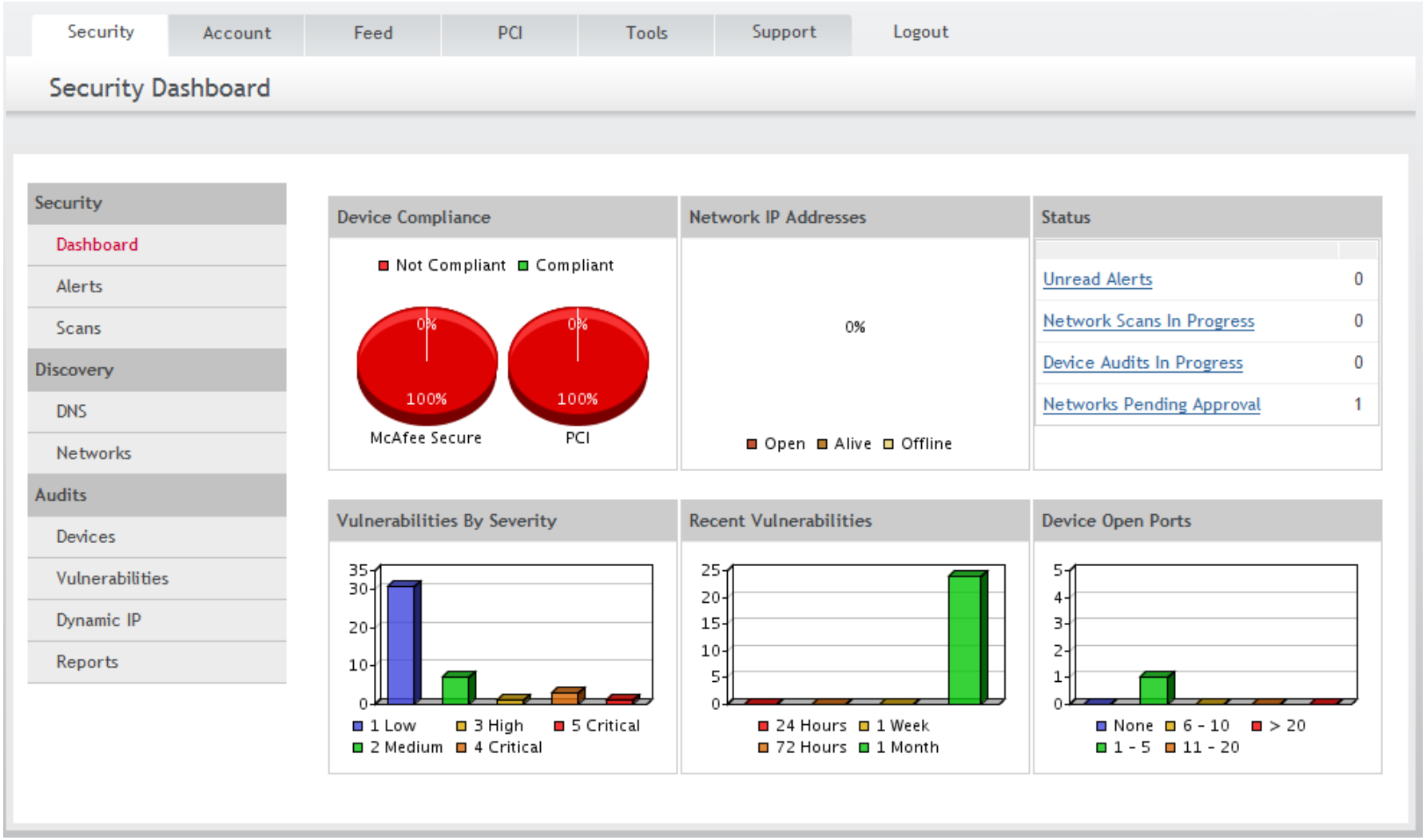
Survey of Web Vulnerability Tools

Local

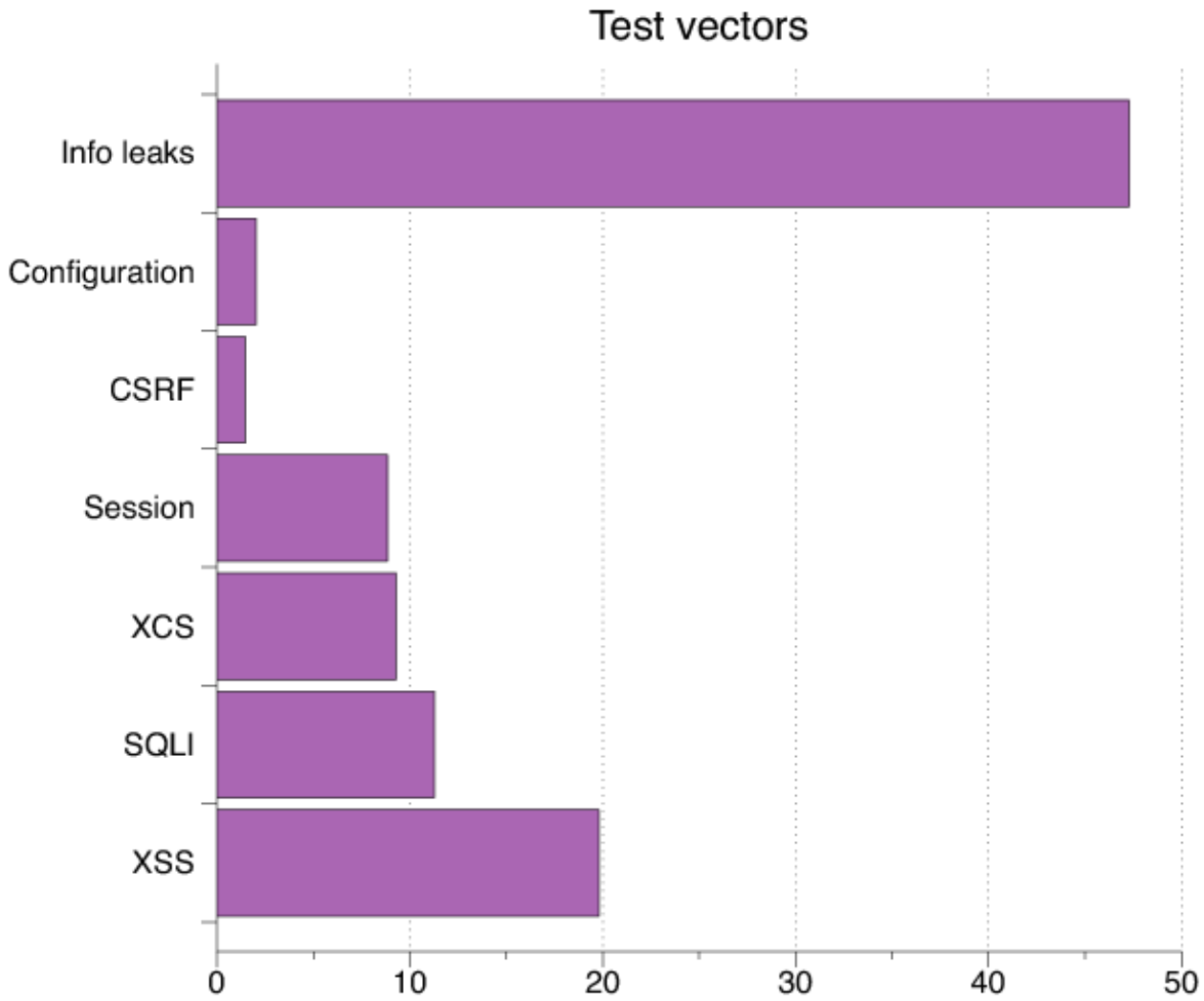
Remote

>$100K total retail price

Example scanner UI



Test Vectors By Category



Test Vector Percentage Distribution

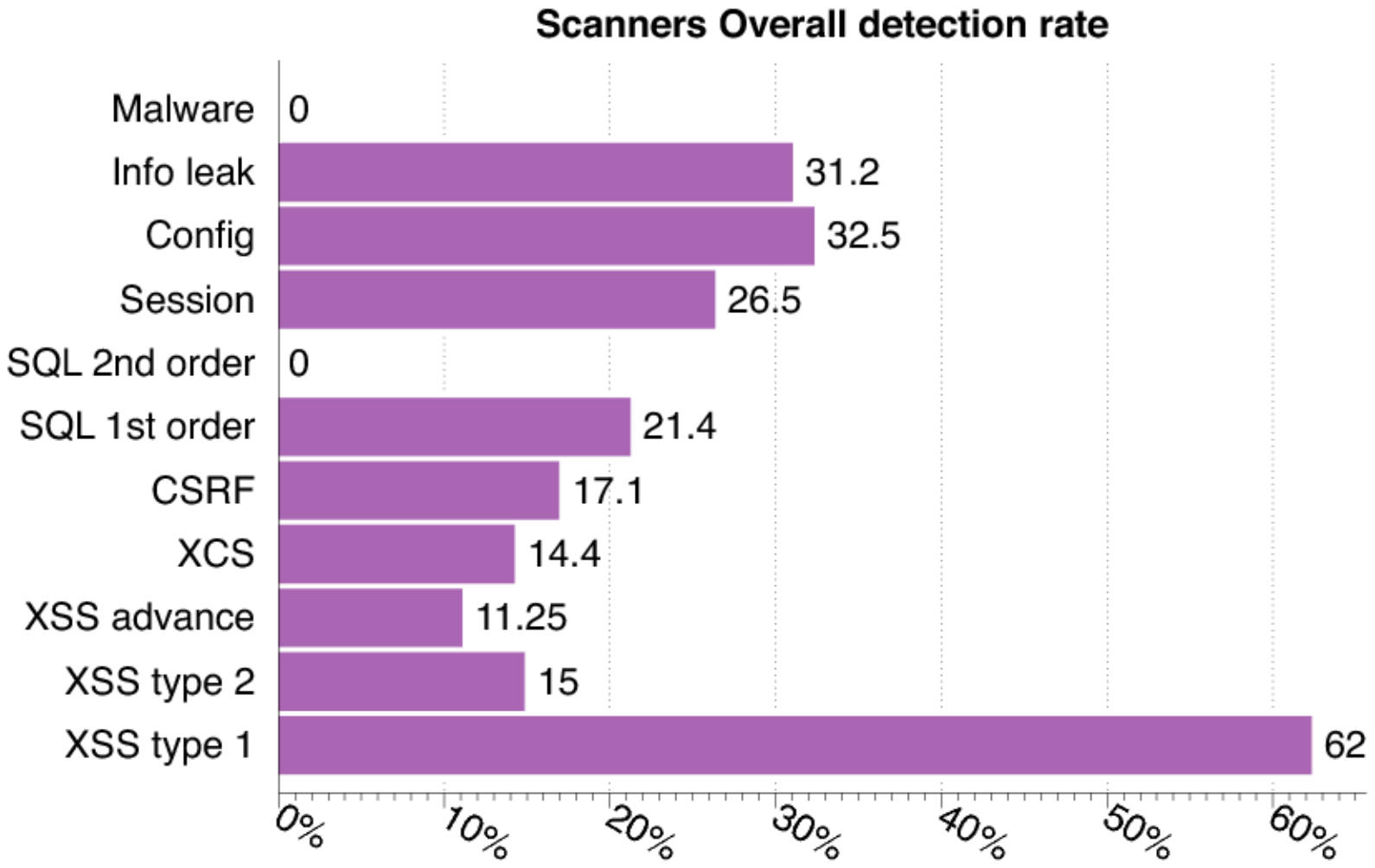
Detecting Known Vulnerabilities

Vulnerabilities for
previous versions of Drupal, phpBB2, and WordPress

Category	Drupal 4.7.0		phpBB2 2.0.19		Wordpress 1.5strayhorn	
	NVD	Scanner	NVD	Scanner	NVD	Scanner
XSS	5	2	4	2	13	7
SQLI	3	1	1	1	12	7
XCS	3	0	1	0	8	3
Session	5	5	4	4	6	5
CSRF	4	0	1	0	1	1
Info Leak	4	3	1	1	5	4

Good: Info leak, Session
Decent: XSS/SQLI
Poor: XCS, CSRF (low vector count?)

Vulnerability Detection



Secure development

Experimental Study

- What factors most strongly influence the likely security of a new web site?
  - Developer training?
  - Developer team and commitment?
    - freelancer vs stock options in startup?
  - Programming language?
  - Library, development framework?
- How do we tell?
  - Can we use automated tools to reliably *measure* security in order to answer the question above?

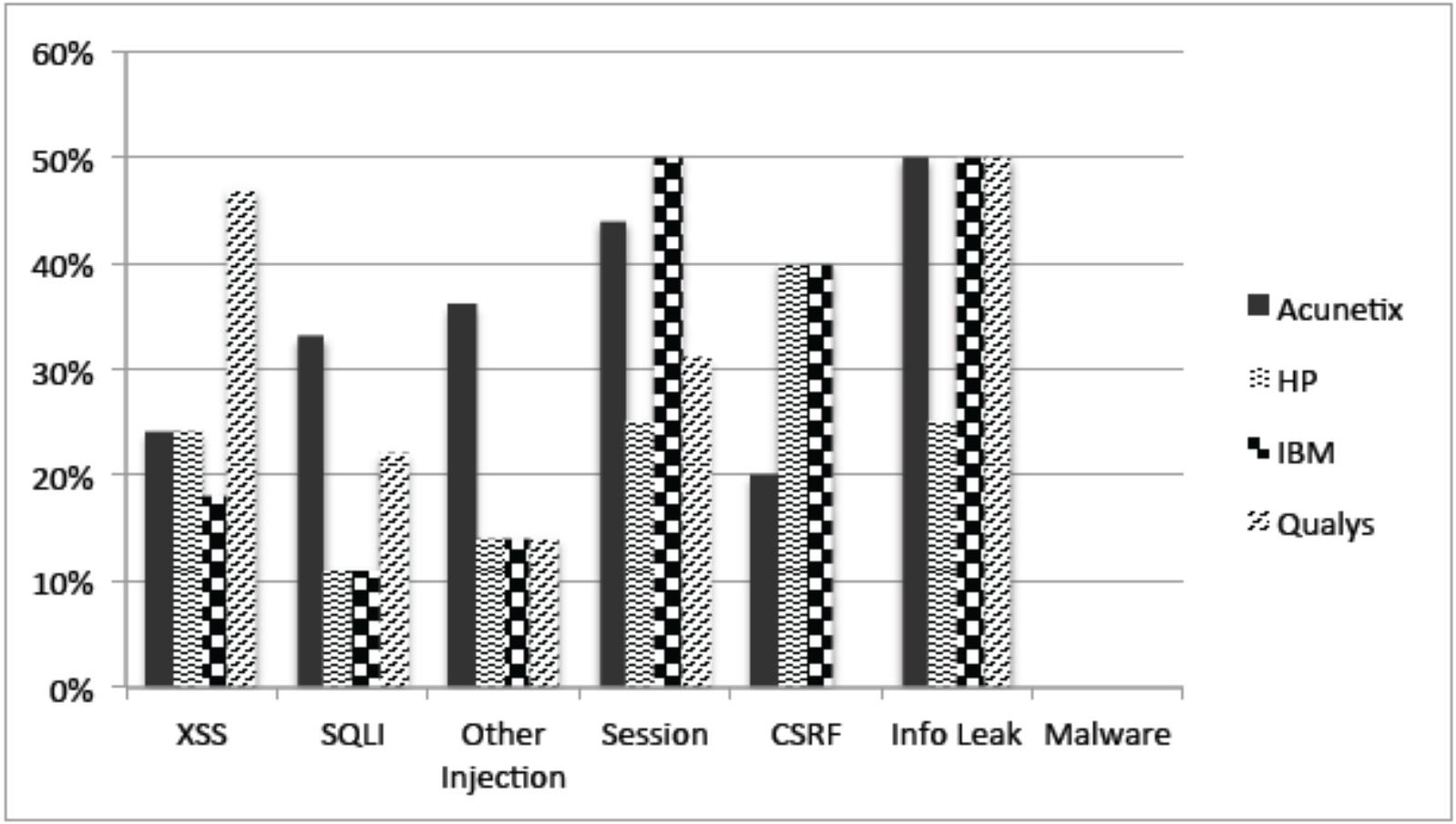
Approach

- Develop a web application vulnerability metric
  - Combine reports of 4 leading commercial black box vulnerability scanners and
- Evaluate vulnerability metric
  - using historical benchmarks and our new sample of applications.
- Use vulnerability metric to examine the impact of three factors on web application security:
  - startup company or freelancers
  - developer security knowledge
  - Programming language framework

Data Collection and Analysis

- Evaluate 27 web applications
  - from 19 Silicon Valley startups and 8 outsourcing freelancers
  - using 5 programming languages.
- Correlate vulnerability rate with
  - Developed by startup company or freelancers
  - Extent of developer security knowledge (assessed by quiz)
  - Programming language used.

Comparison of scanner vulnerability detection

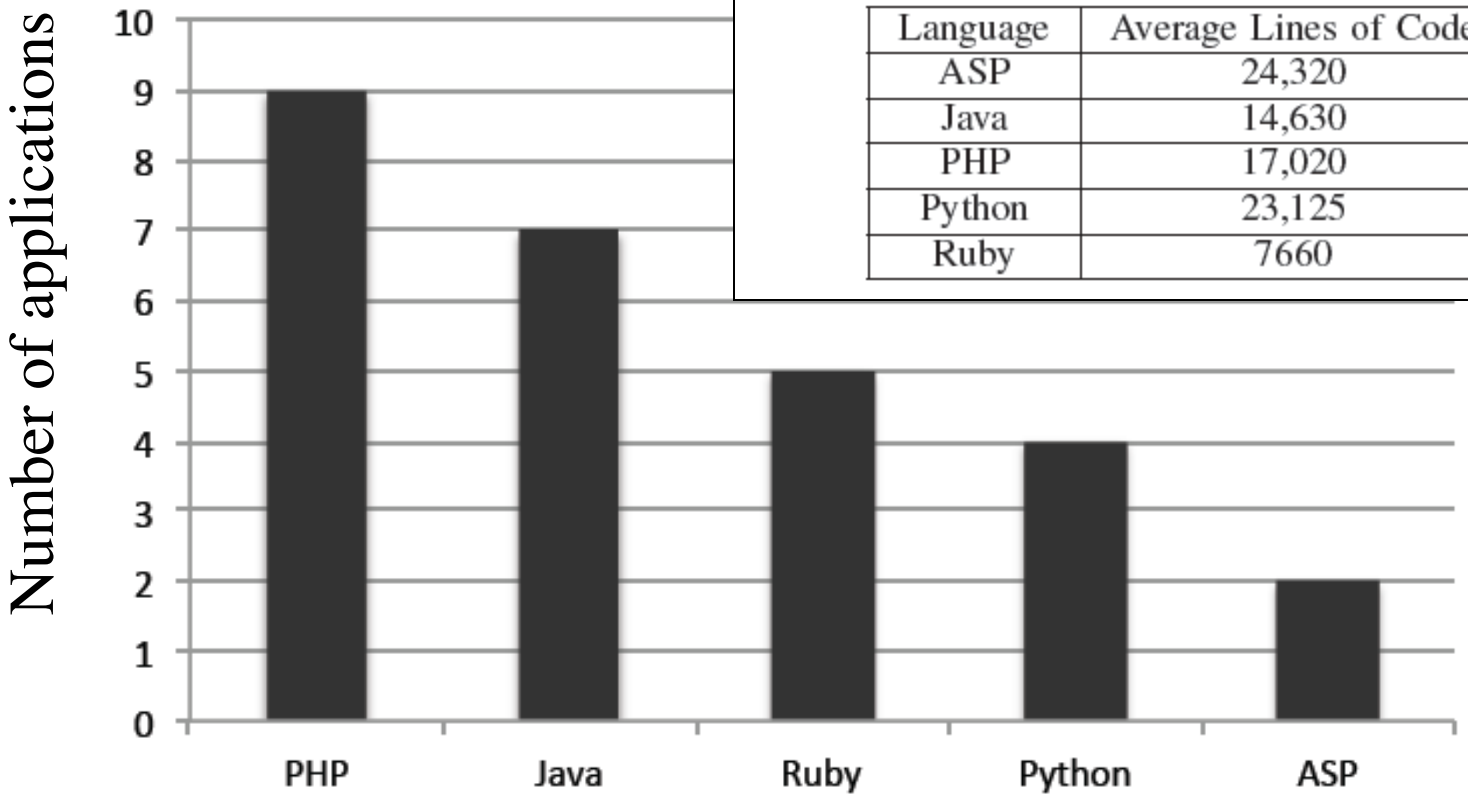


Developer security self-assessment

QUIZ CATEGORIES AND QUESTION SUMMARY

Q	Category Covered	Summary
1	SSL Configuration	Why CA PKI is needed
2	Cryptography	How to securely store passwords
3	Phishing	Why SiteKeys images are used
4	SQL Injection	Using prepared statements
5	SSL Configuration/XSS	Meaning of "secure" cookies
6	XSS	Meaning of "httponly" cookies
7	XSS/CSRF/Phishing	Risks of following emailed link
8	Injection	PHP local/remote file-include
9	XSS	Passive DOM-content intro. methods
10	Information Disclosure	Risks of auto-backup (~) files
11	XSS/Same-origin Policy	Consequence of error in Applet SOP
12	Phishing/Clickjacking	Risks of being iframed

Language usage in sample



AVERAGE LINES OF CODE FOR EACH LANGUAGE

Language	Average Lines of Code
ASP	24,320
Java	14,630
PHP	17,020
Python	23,125
Ruby	7660

Summary of developer study

- Security scanners are useful but not perfect
  - Tuned to current trends in web application development
  - Tool comparisons performed on single testbeds are not predictive in a statistically meaningful way
  - Combined output of several scanners is a reasonable comparative measure of code security, compared to other quantitative measures
- Based on scanner-based evaluation
  - Freelancers are more prone to introducing injection vulnerabilities than startup developers, in a statistically meaningful way
  - PHP applications have statistically significant higher rates of injection vulnerabilities than non-PHP applications; PHP applications tend not to use frameworks
  - Startup developers are more knowledgeable about cryptographic storage and same-origin policy compared to freelancers, again with statistical significance.
  - Low correlation between developer security knowledge and the vulnerability rates of their applications

Warning: don't hire freelancers to build secure web site in PHP.

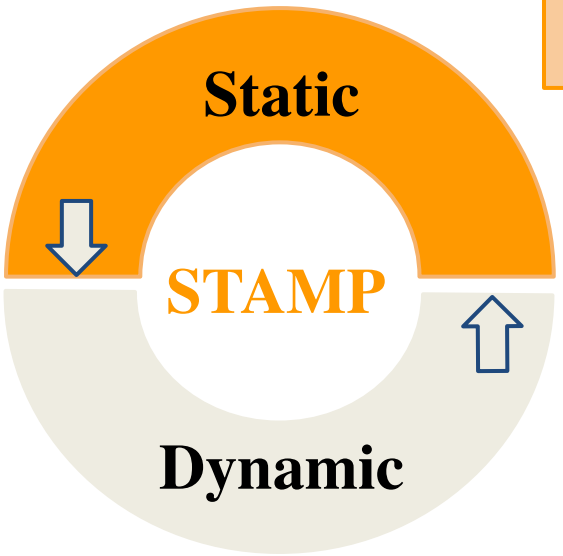
Outline

- General discussion of code analysis tools
  - Goals and limitations of static, dynamic tools
  - Static analysis based on abstract states
- Security tools for traditional systems programming
  - Property checkers from Engler et al., Coverity
  - Sample security-related results
- Web security analysis
  - Black-box security tools
  - Study based on these tools: security of coding
- Static analysis for Android malware
  - Determining whether app is malicious
  - Using tools for other security studies

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

STAMP Admission System

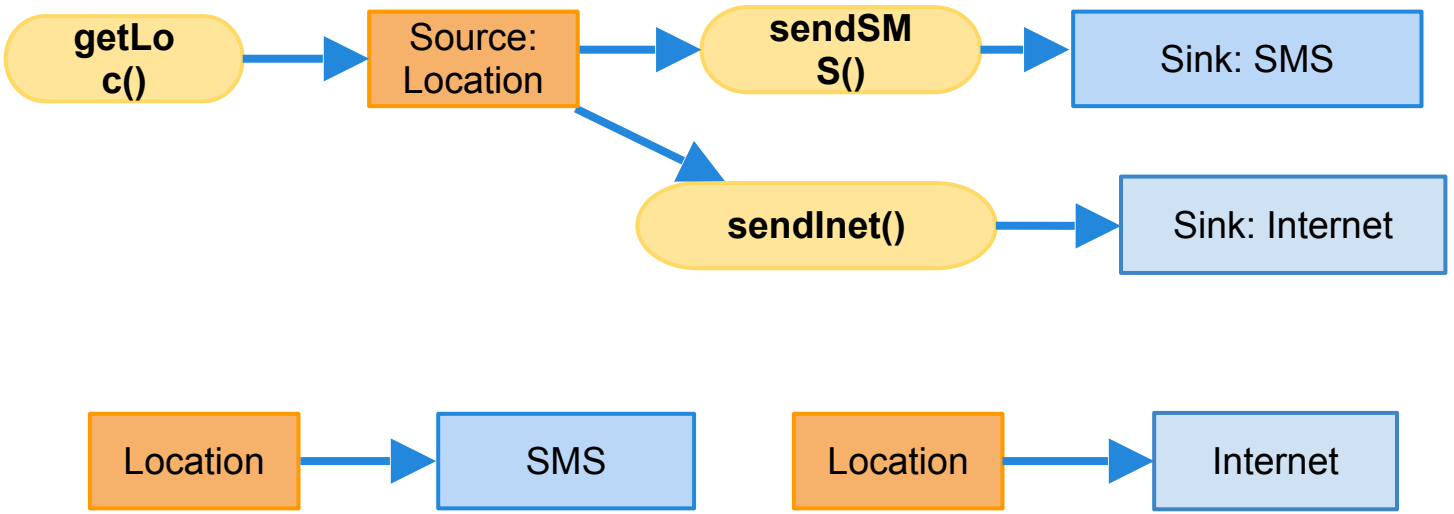
Static Analysis
More behaviors,
fewer details



Dynamic Analysis
Fewer behaviors,
more details

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

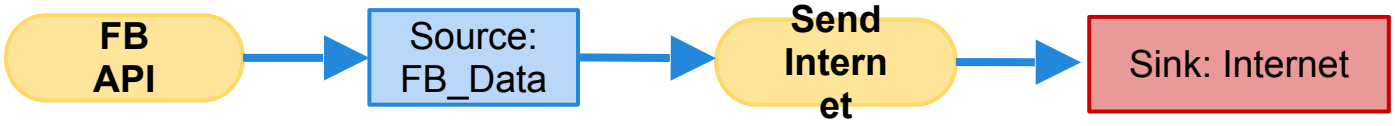
Data Flow Analysis



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

Applications of Data Flow Analysis

- 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

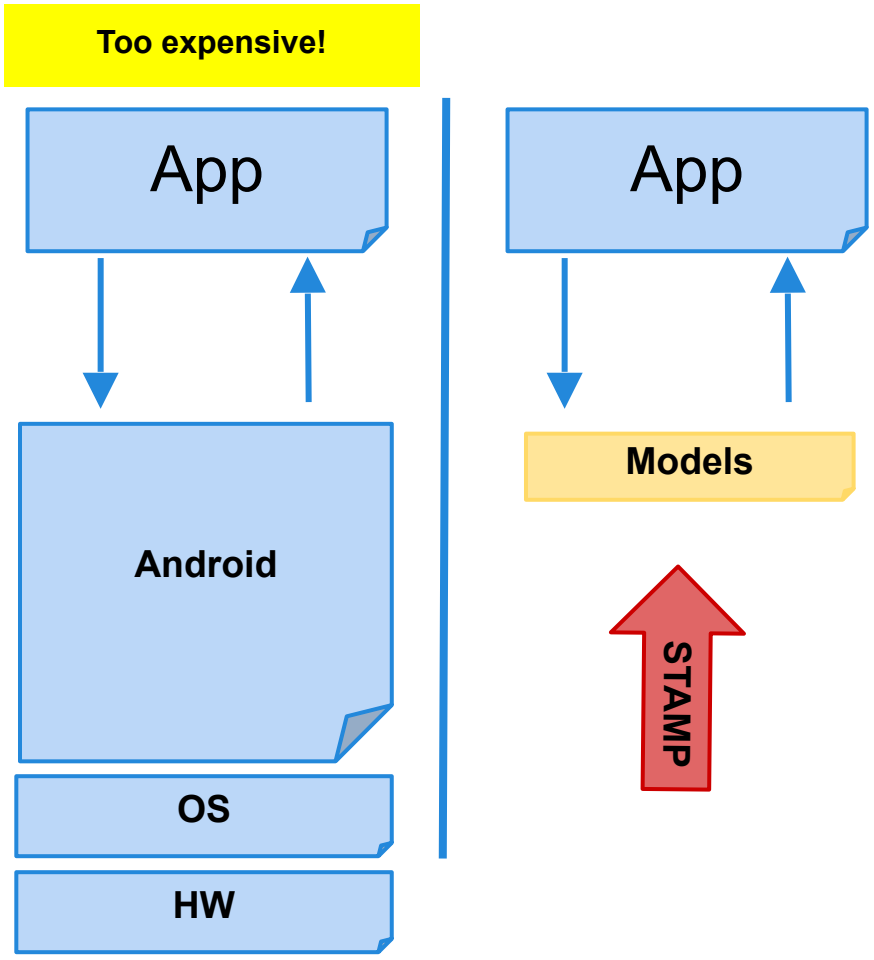
Privacy Policy
This app collects your:
Contacts
Phone Number
Address

- 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

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

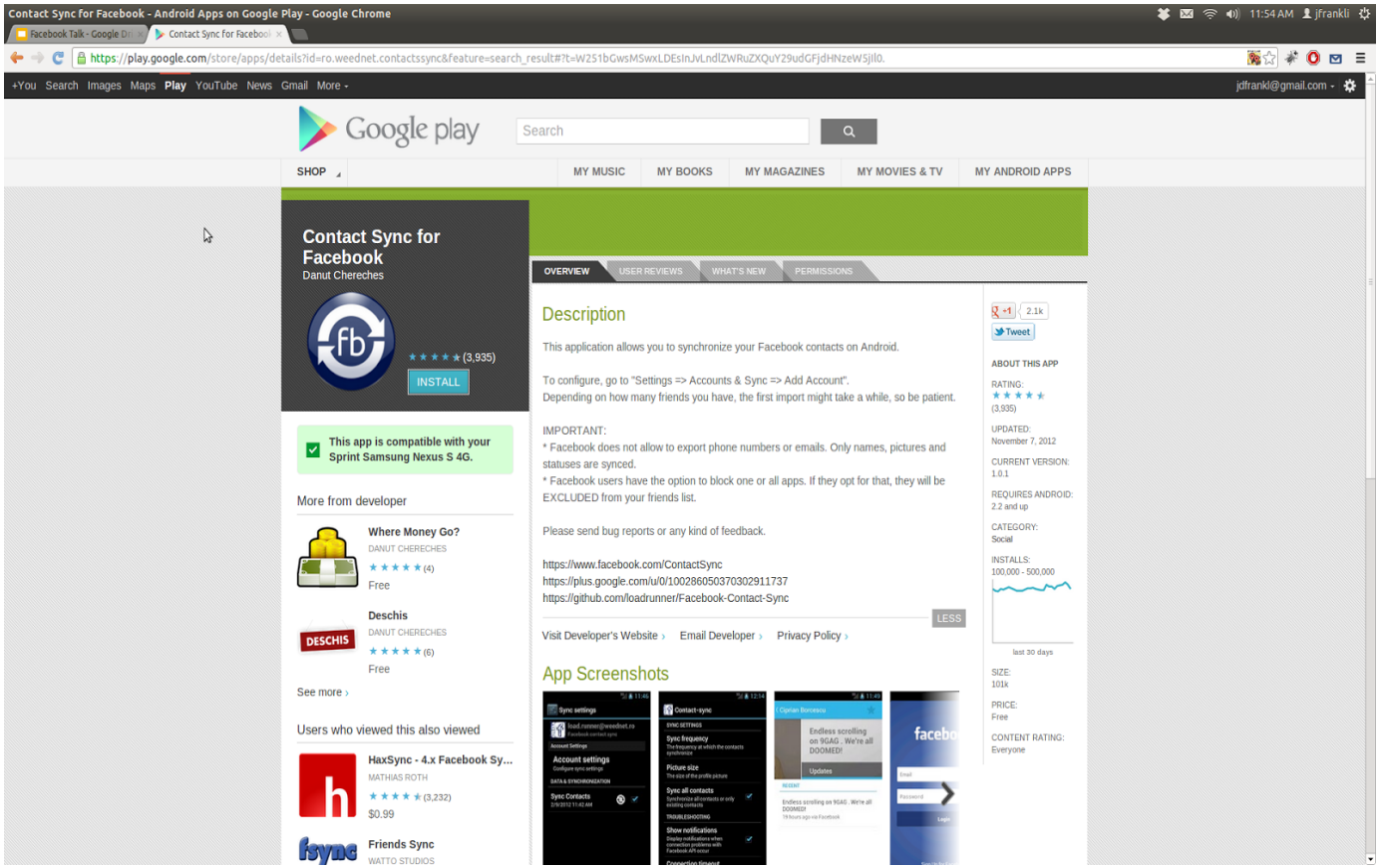
Currently Detectable Flow Types

396 Flow Types

Unique Flow Types = Sources x Sink

Example Analysis

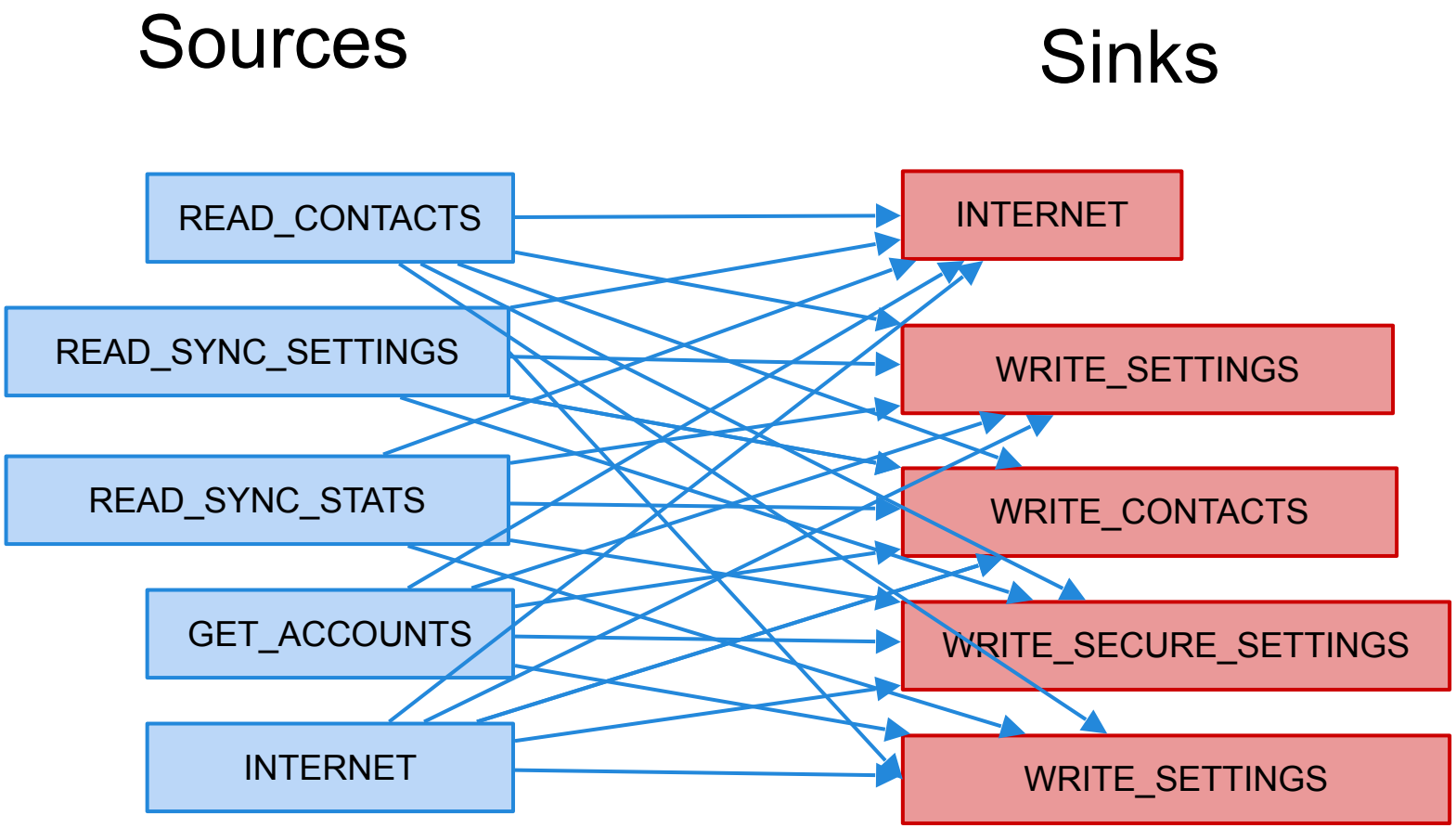
Contact Sync for Facebook (unofficial)



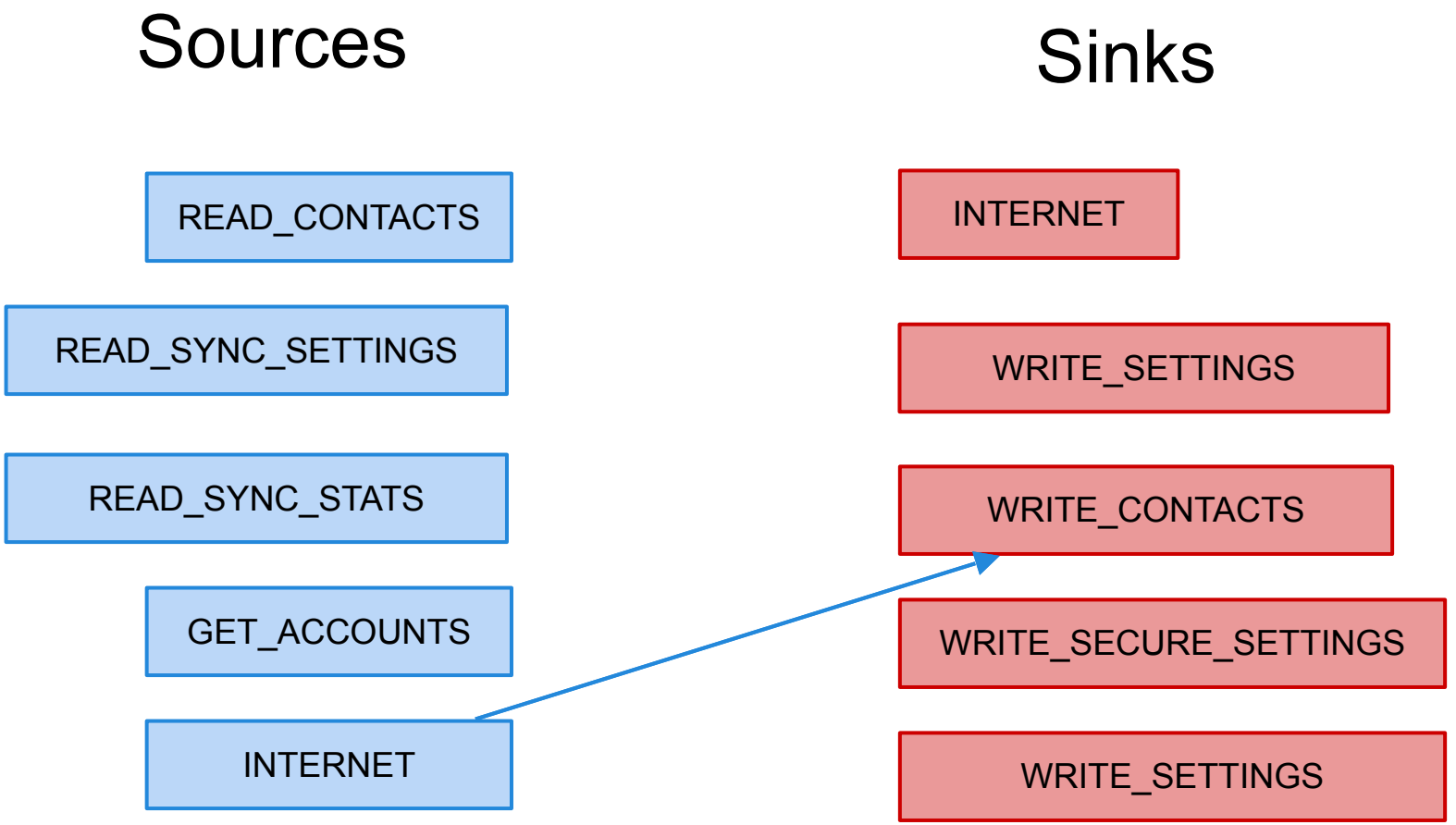
Contact Sync Permissions

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

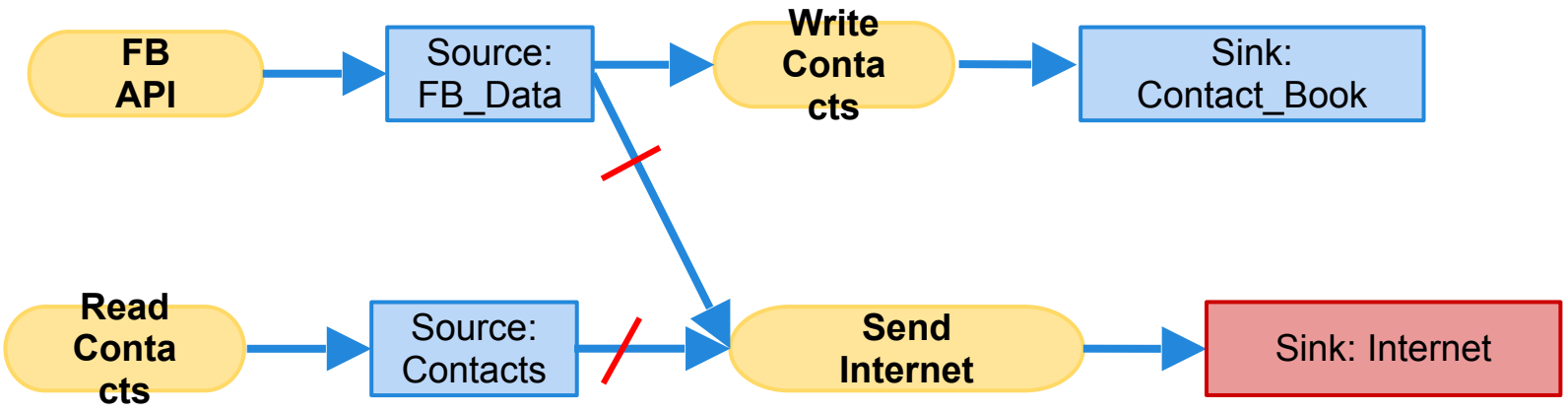
Possible Flows from Permissions



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()

Sample results

Analyze 998,286 free web apps from June 2014

Mobile Web App Feature	% Apps
JavaScript Enabled	97
JavaScript Bridge	36
shouldOverrideUrlLoading	94
shouldInterceptRequest	47
onReceivedSslError	27
postUrl	2
Custom URL Patterns	10

Vuln	% Relevant	% Vulnerable
Unsafe Navigation	15	34
Unsafe Retrieval	40	56
Unsafe SSL	27	29
Exposed POST	2	7
Leaky URL	10	16

Summary

- General discussion of code analysis tools
  - Goals and limitations of static, dynamic tools
  - Static analysis based on abstract states
- Security tools for traditional systems programming
  - Property checkers from Engler et al., Coverity
  - Sample security-related results
- Web security analysis
  - Black-box security tools
  - Study based on these tools: security of coding
- Static analysis for Android malware
  - Determining whether app is malicious
  - Using tools for other security studies

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