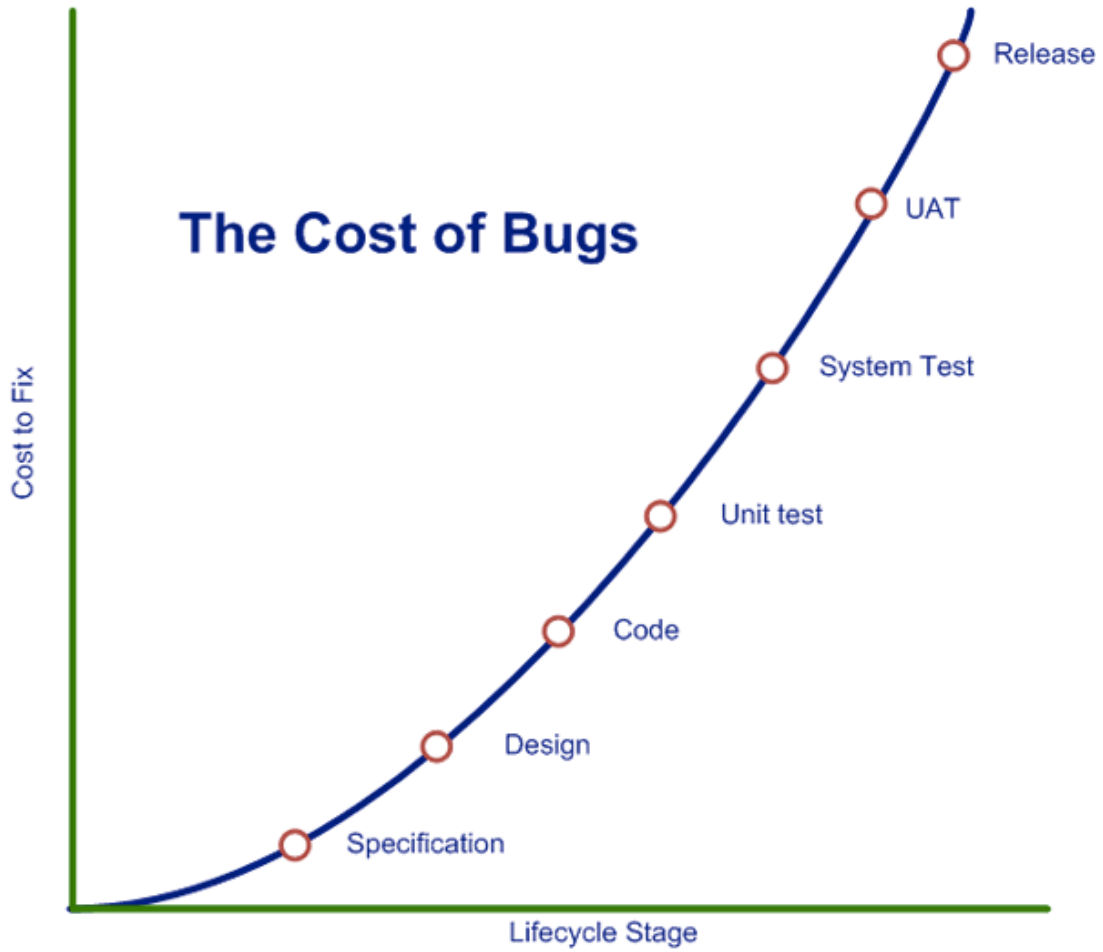
ROBUST AND SECURE APPLICATION CODE THROUGH ANALYSIS

Dr. Benjamin Livshits

Cost of Fixing a Defect



How Do We Find Bugs?

Static Analysis

- Static code analysis toos
  - Coverity
  - Tools from Microsoft

    like Prefix and Prefast
  - FindBugs (for Java)
  - Fortify (for security)

- **Pros**:
  - Near-perfect code coverage, exercise all paths
  - Can be run, incrementally as part of development process
- **Cons**:
  - Can be imprecise
  - Can scale poorly
  - Can produce results that are tough to interpret

Runtime Monitoring

- Instrument code for testing
- Heap memory: Purify
- Valgrind: http://valgrind.org
- Perl tainting (information flow)
- Java race condition checking

- **Pros**:
  - Easy to reproduce the bug
  - Relatively easy to implement
- **Cons**:
  - Slows down the program significantly
  - 10x-40x slowdowns
  - Test only: cannot be used in production
  - Not all paths executed

Black-box Testing

- Fuzzing and **penetration testing**
- Black-box web application security analysis
- Typically, tries to provide cleverly crafted unexpected inputs
- Also knows as *inputs of death*
- Example:
  - Peach fuzzer http://peachfuzzer.com/
  - antifuzzer, Dfuz, SPIKE, GPF, etc.

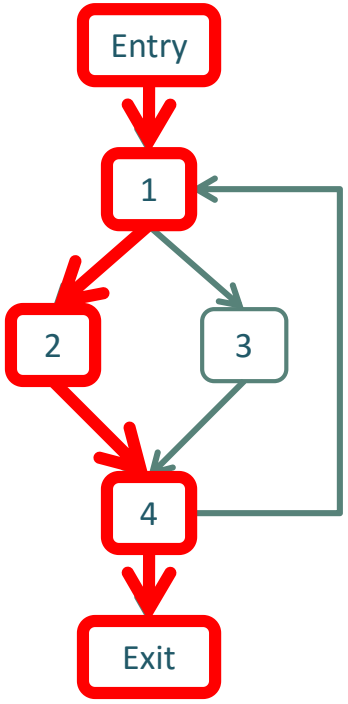
- **Pros:**
  - Easy to reproduce the bug
  - Don't need to understand the code
  - Can be done by someone else
- **Cons:**
  - Have no visibility into program logic
  - Has low coverage
  - Possibly lots of missing vulnerabilities

Outline

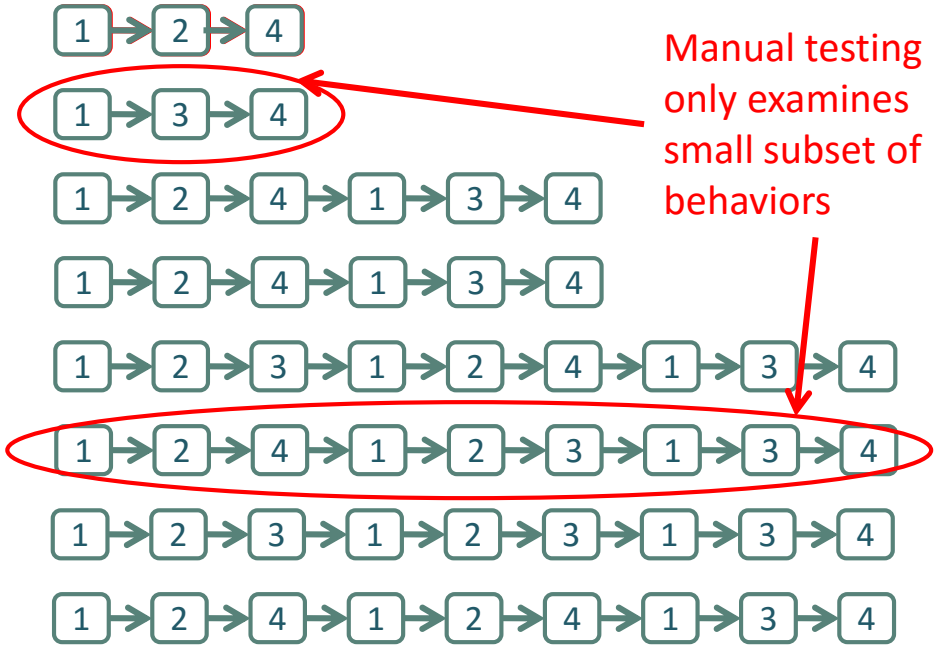
- General discussion of static analysis tools
  - Goals and limitations
  - Approach based on abstract states
- More about one specific approach
  - Property checkers from Engler et al., Coverity
  - Sample security-related results

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

Static Analysis Coverage Advantage

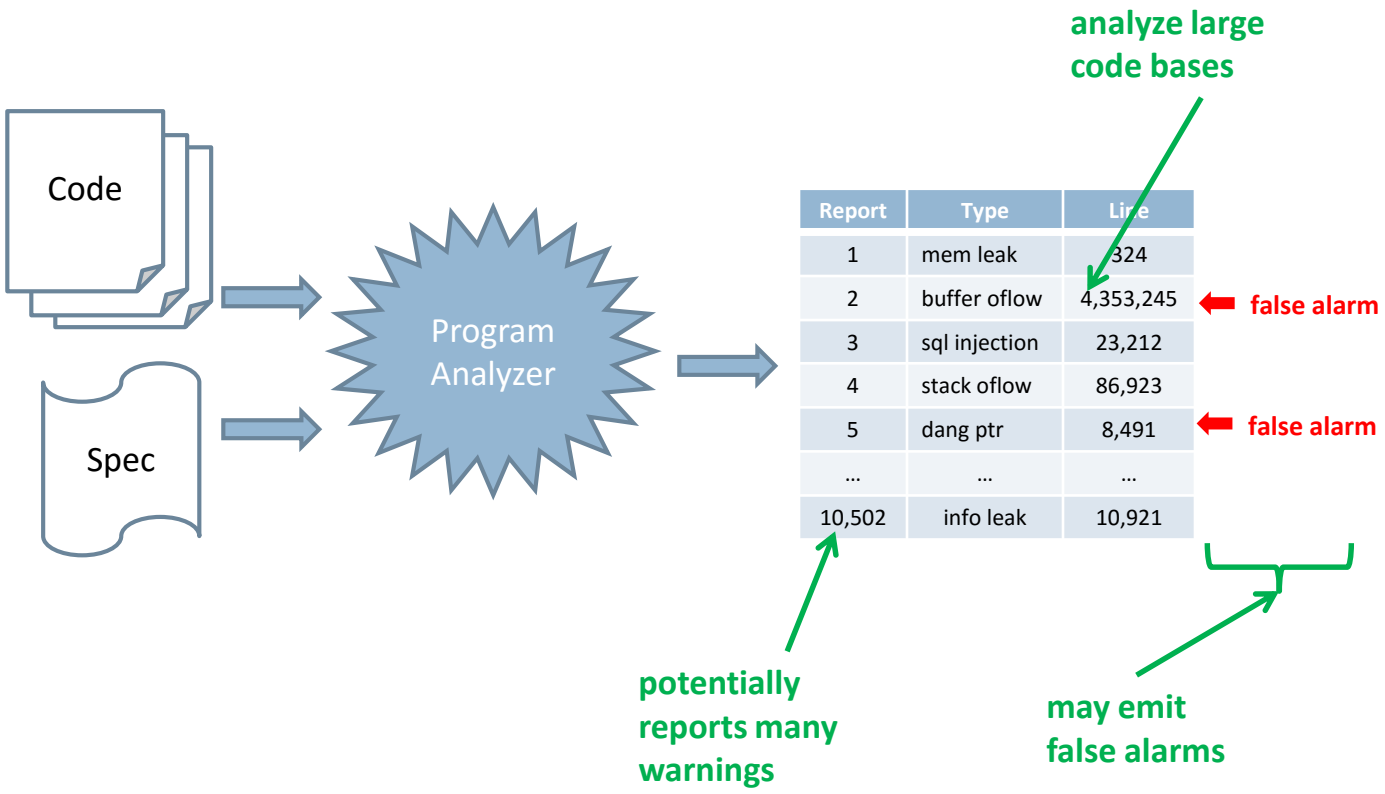


Software



Behaviors

Program Analyzers



Static Analysis Goals

- **Bug finding**: identify code that the programmer wishes to modify or improve

- **Correctness**: *Verify* the absence of certain classes of errors

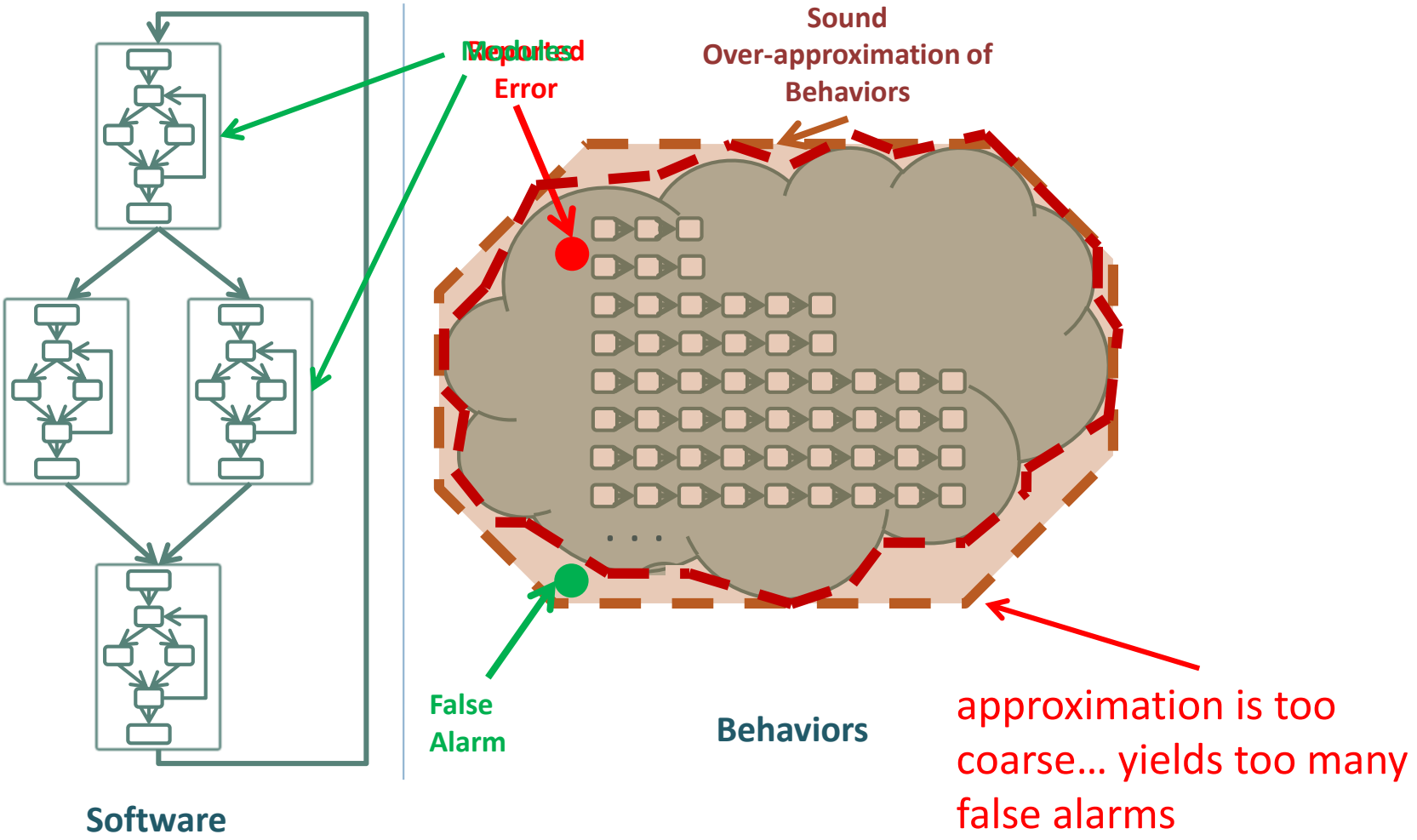
Soundness and Completeness

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

Decidable?

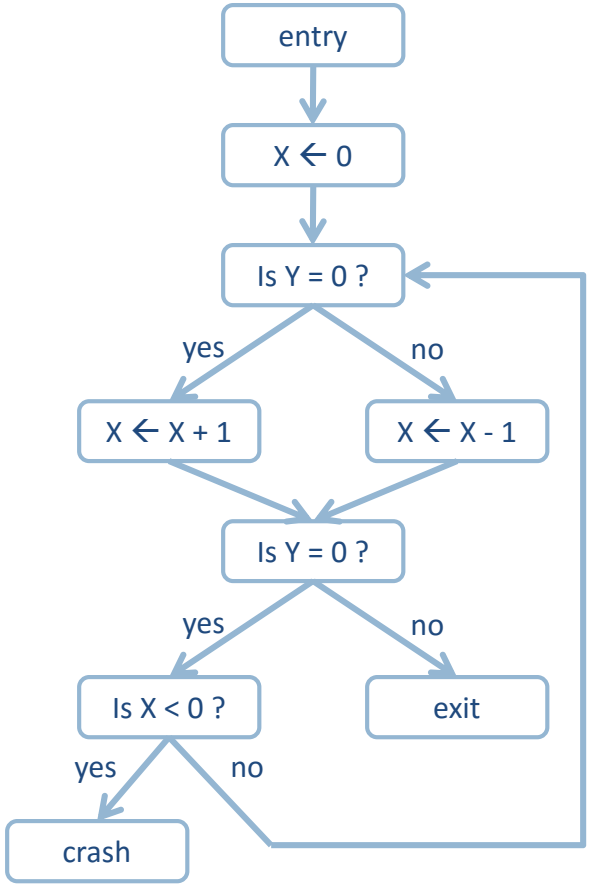
	Complete	Incomplete
Sound	Reports all errors Reports no false alarms **Undecidable**	Reports all errors May report false alarms **Decidable**
Unsound	May not report all errors Reports no false alarms **Decidable**	May not report all errors May report false alarms **Decidable**

Over- and Underapproximations

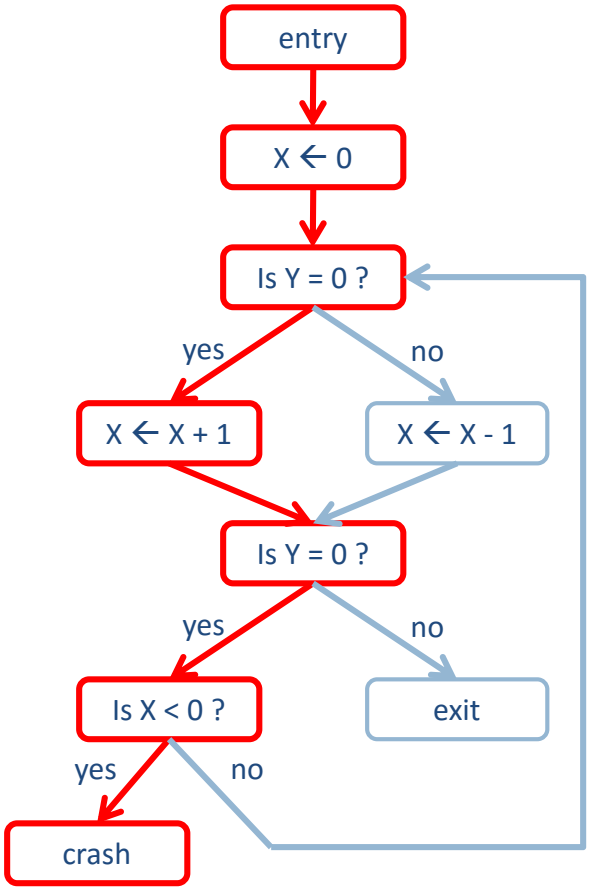


Does This Program Ever Crash?

```
x=0;

L: if(y==0){
   x=x+1;
}else{
   x = x-1;
}

if(y==0){
  if(x<0){
    crash()
  }else{
    goto L;
  }
}
```

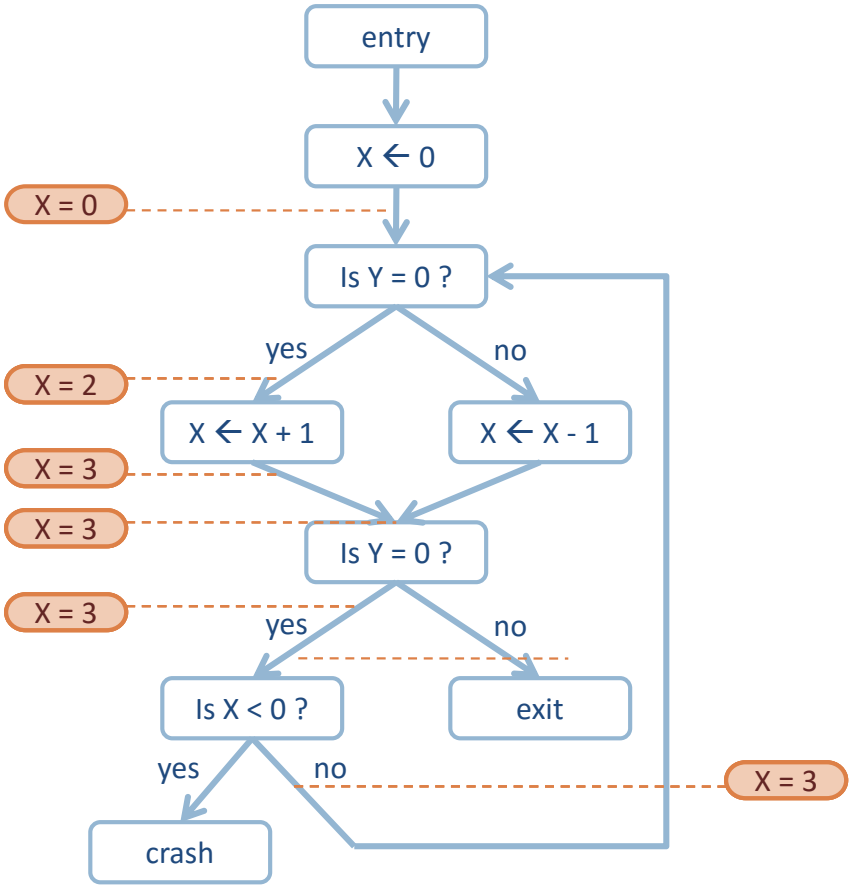


Does This Program Ever Crash?



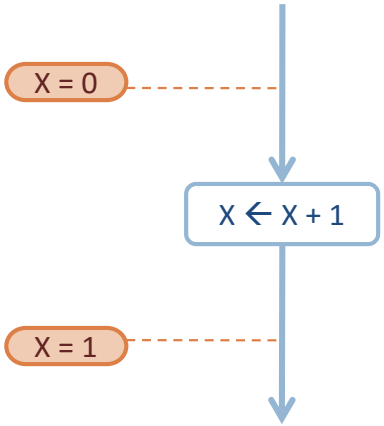
infeasible path!
overly imprecise
… program will never crash

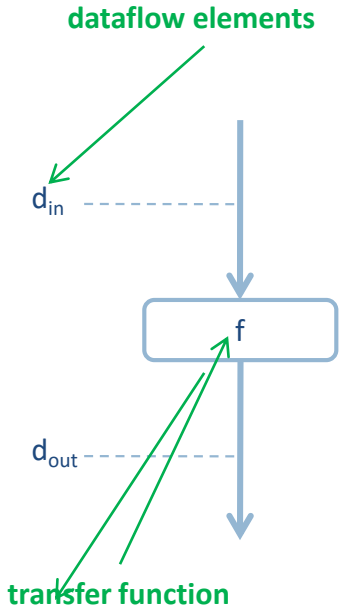
Try Analyzing Without Approximation



non-termination!
… therefore, need to approximate

Dataflow Analysis Framework

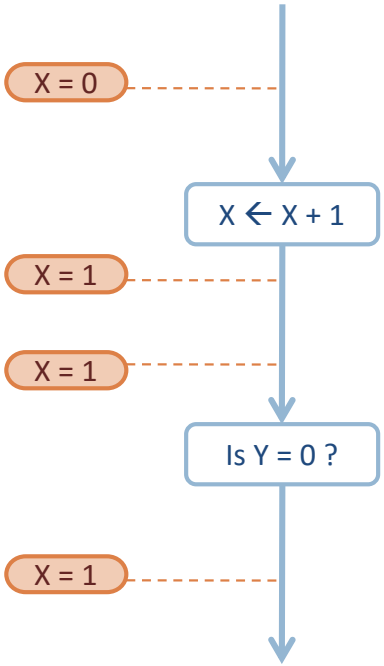


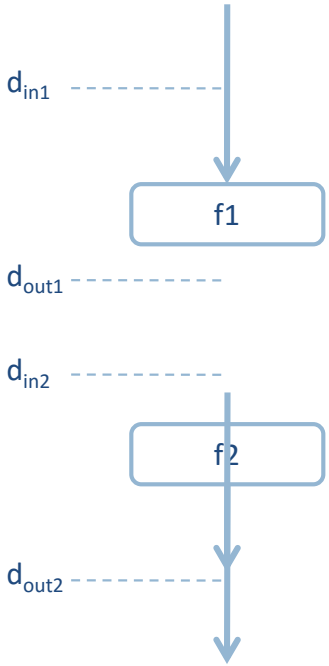


$$d_{out} = f(d_{in})$$

dataflow equation

Applying the Dataflow Approach



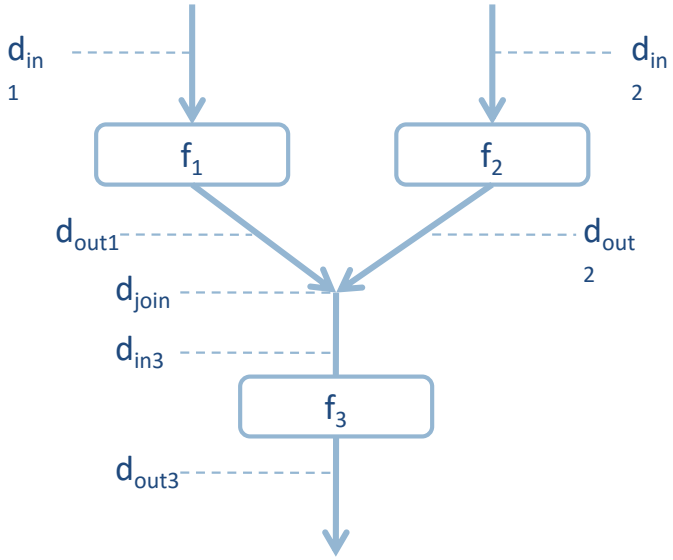


$$d_{out1} = f_1(d_{in1})$$

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

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

Meet/Join Operator ⊔



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

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

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

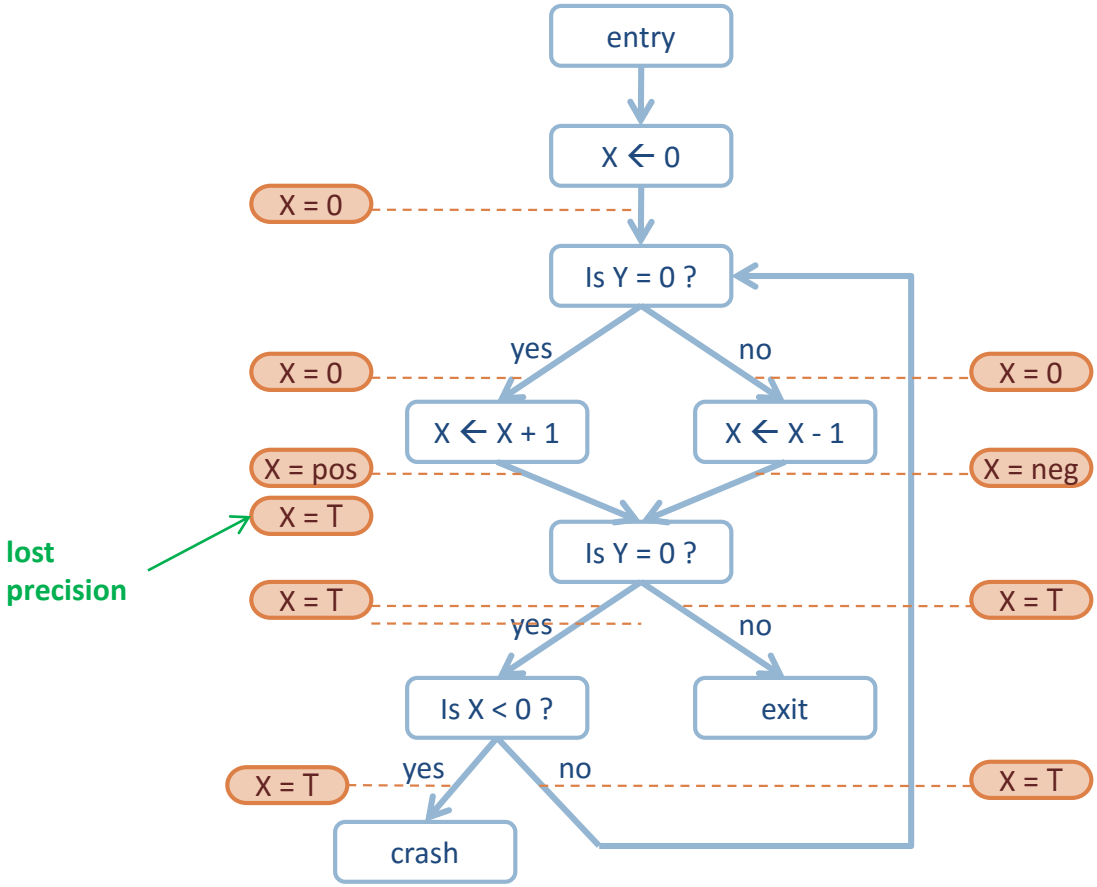
$$d_{join} = d_{in3}$$

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

least upper bound operator
Example: union of possible values

What is the space of dataflow elements, Δ?
What is the least upper bound operator, ⊔?

Try Analyzing with "Signs" Approximation...

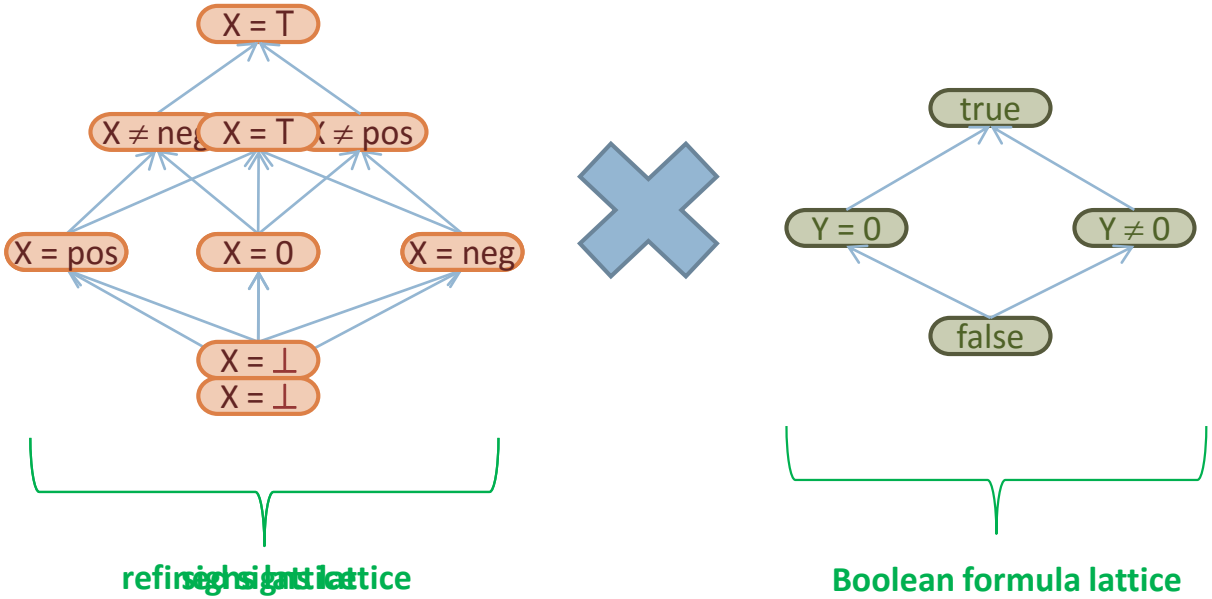


terminates...

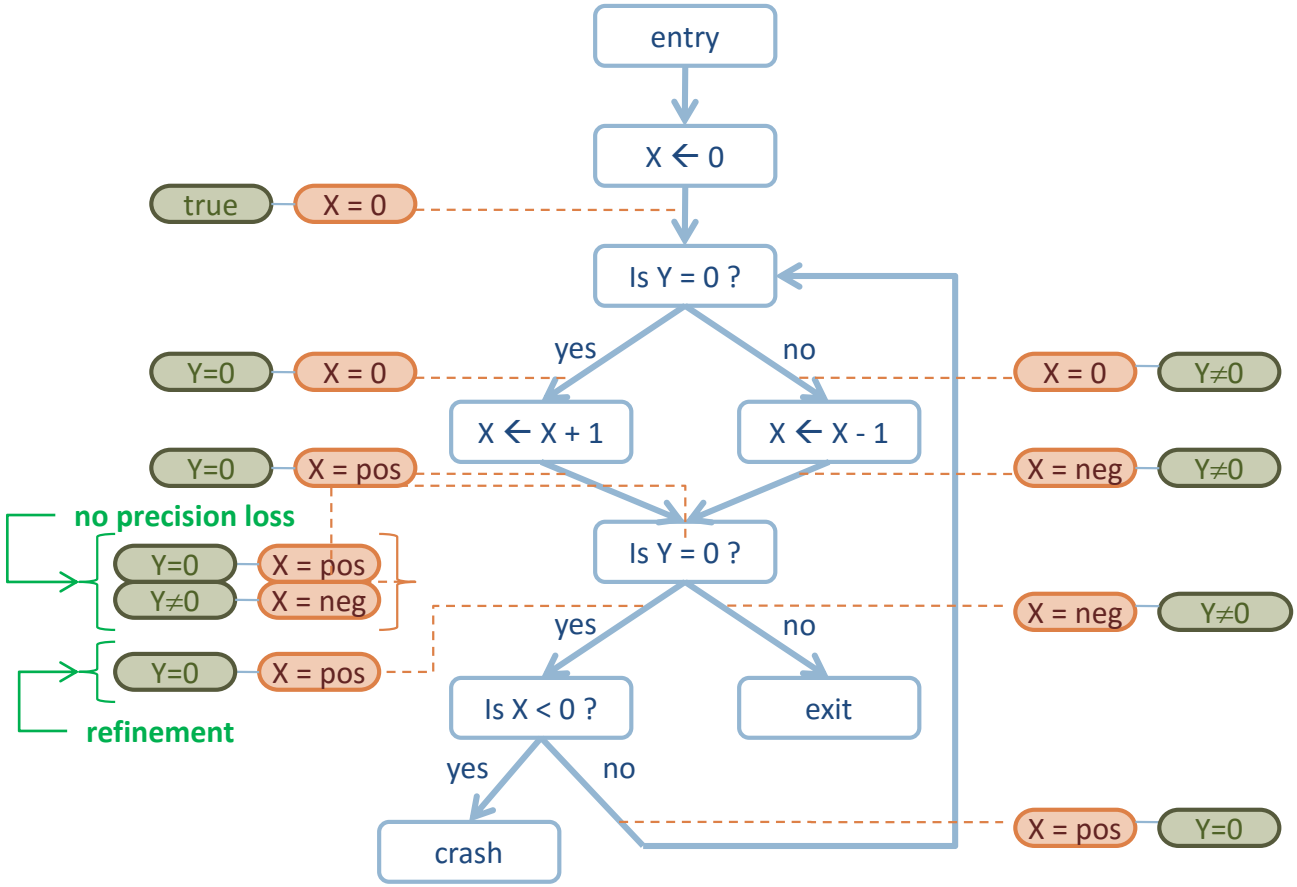
... but reports false alarm

... therefore, need more precision

Lattices


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

Try Analyzing with “Path-sensitive signs”



terminates...

… no false alarm

… soundly proved never crashes

Bugs Detected by Coverity

- 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
- Under-allocations of dynamic data
- Memory leaks
- File handle leaks
- Network resource leaks
- Unused values
- Unhandled return codes
- Use of invalid iterators

From Coverity

1. Condition `server != null`, taking true branch

2. **alloc_fn:** A new resource is returned from allocation method `getZooKeeper`. (The virtual call resolves to `org.apache.hadoop.hbase.master.cleaner.TestHFileCleaner.DummyServer.getZooKeeper`.)

3. Condition `server.getZooKeeper() != null`, taking true branch

CID 1090416 (#1 of 2): Resource leak (RESOURCE_LEAK)

4. **leaked_resource:** Failing to save or close resource created by `server.getZooKeeper()` leaks it.

```
302   if (server != null && server.getZooKeeper() != null) {
303     try {
304       createNodeSplitting(s
305         parent.getRegionInf
306     } catch (KeeperExceptio
307       throw new IOException
308         this.parent.getRegi
309     }
310   }
311   this.journal.add(JournalE
```

```
@Override
public ZooKeeperWatcher getZooKeeper() {
  try {
    return new ZooKeeperWatcher(getConfiguration(), "dummy server", this);
  } catch (IOException e) {
    e.printStackTrace();
  }
  return null;
}
```

Example Checker: Missing Optional Arguments

- Prototype for `open()` syscall:

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

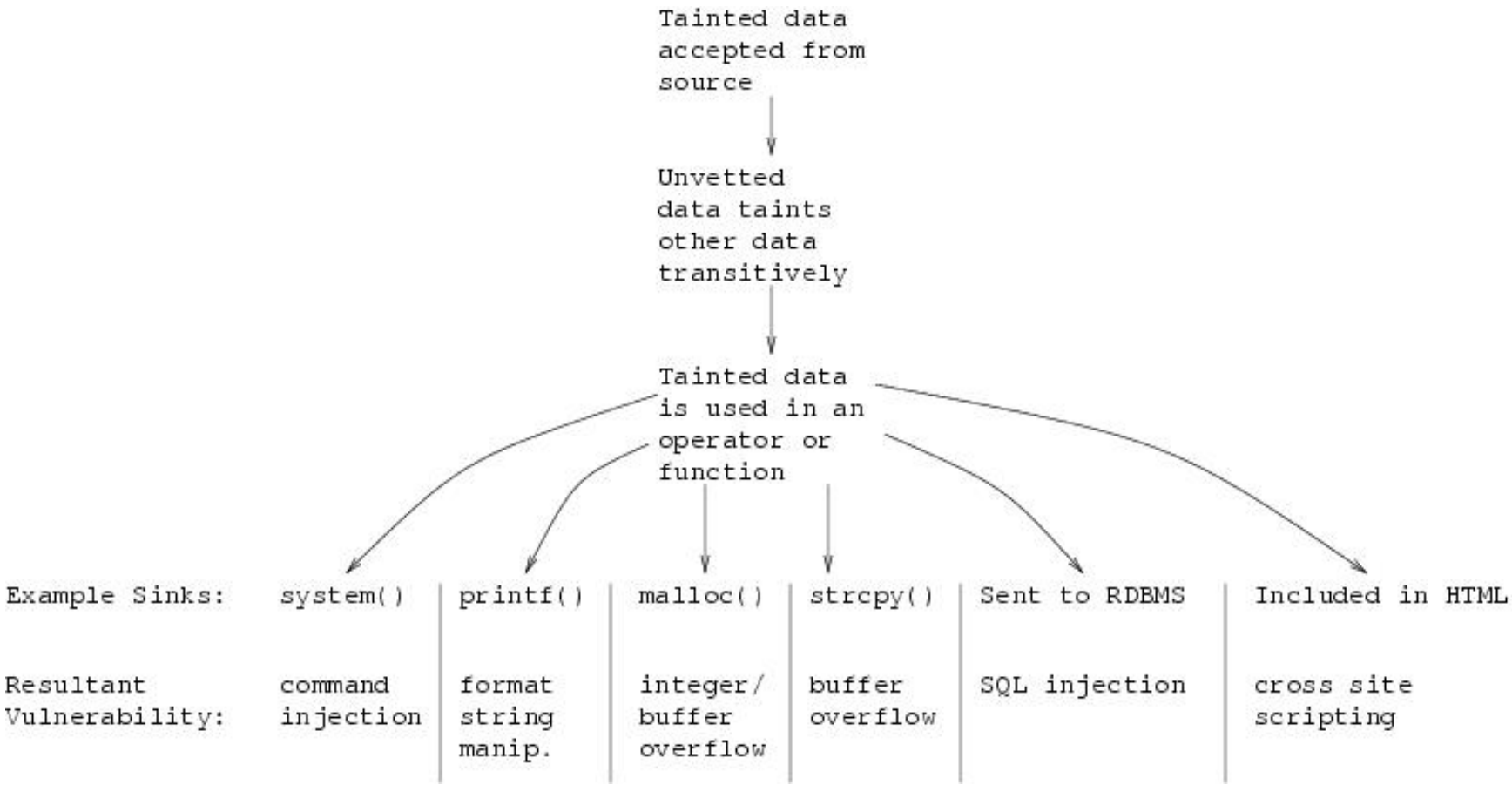
- Typical mistake:

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

- Force setting explicit file **permissions**!

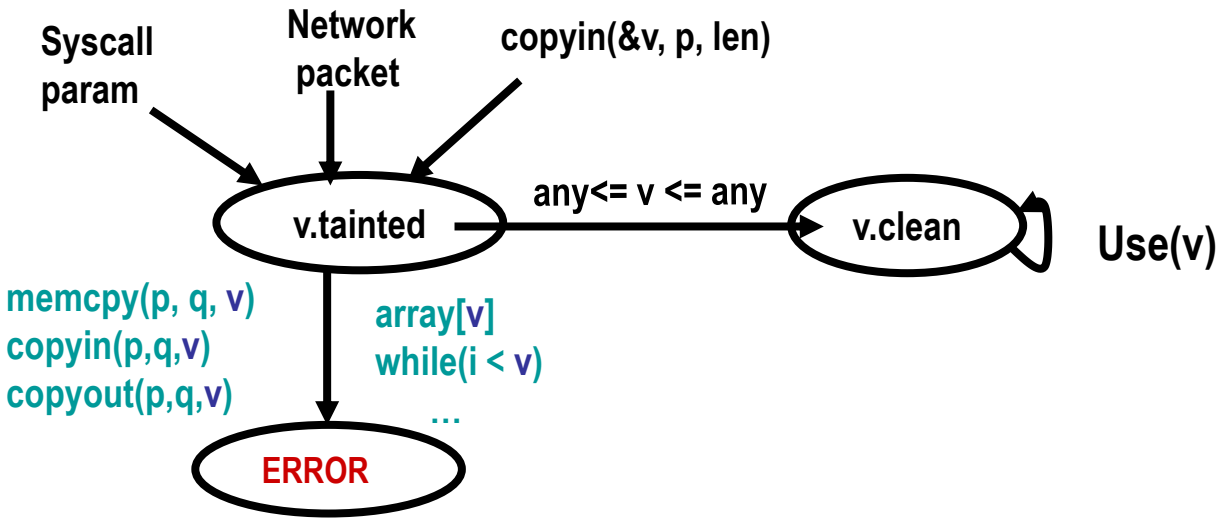
- Check: Look for `oflags == O_CREAT` without mode argument

Tainting Checkers



Sanitize Integers Before Use

Warn when unchecked integers from untrusted sources reach trusting sinks



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

Looking for Blocking Function Calls


```
204 };
205 START_EXTEND_CHECKER( block_check, int_store );
206 ANALYZE_TREE()
207 {
208     Fun locking_fun("lock");
209     Fun unlocking_fun("unlock");
210     Fun blocking_fun("fopen");
211     if ( MATCH(locking_fun) ) {
212         SET_STATE(LOCKED);
213     } else if ( MATCH(unlocking_fun) ) {
214         SET_STATE(UNLOCKED);
215     } else if ( MATCH(blocking_fun) ) {
216         if ( GET_STATE() == LOCKED ) ) {
217         COMMIT_ERROR("Called fopen within a locking context.");
218         }
```

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);
```

- `d` is read from the user
- Signed integer `d.idx` is upper-bound checked but not lower-bound checked
- `d.used` is *unchecked*, allowing 2GB of user data to be copied into the kernel

Remote Exploit


```
/* 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);
```

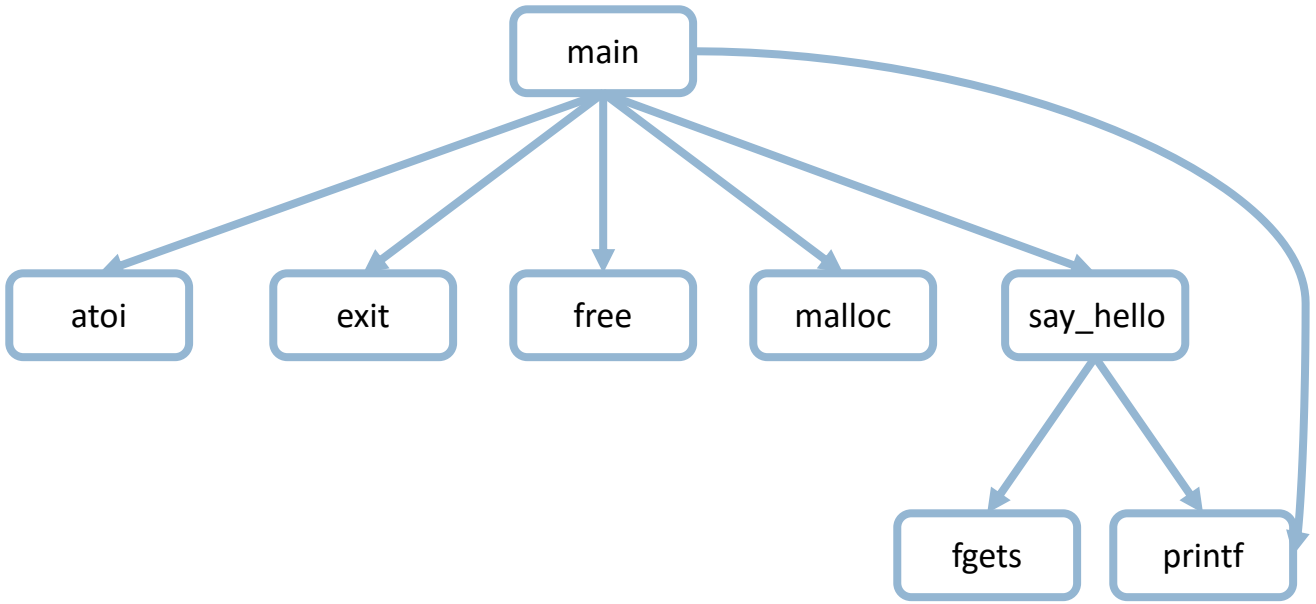
- `msg` points to arbitrary network data
- This can be used to overflow `cmd` and write data onto the stack

Example Code with Functions and Calls

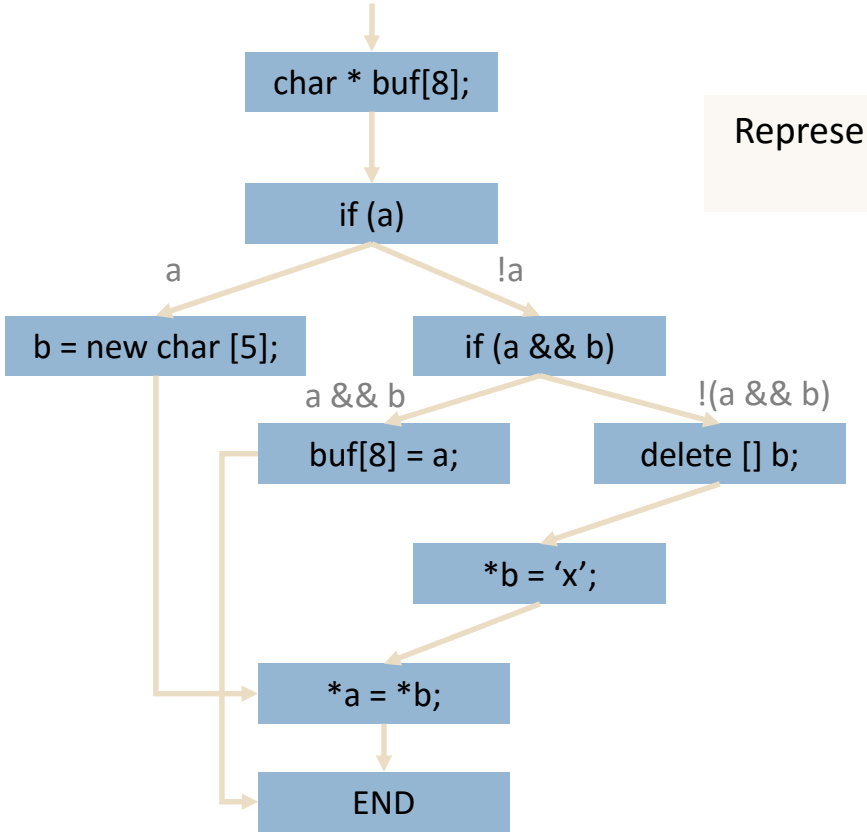

```
#include <stdlib.h>
#include <stdio.h>

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);
  }
}
```

- We would want to reason about the flow of the input (**size**) and **name** provided by the user

Call Graph for the Program

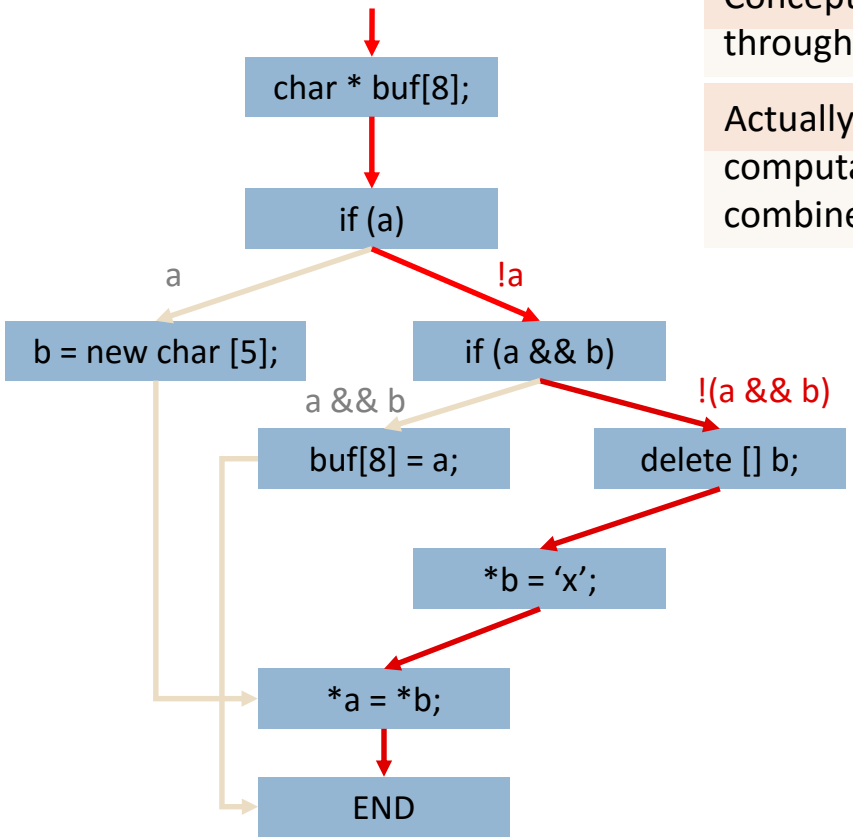


Control Flow Graph



Represent logical structure of code in graph form

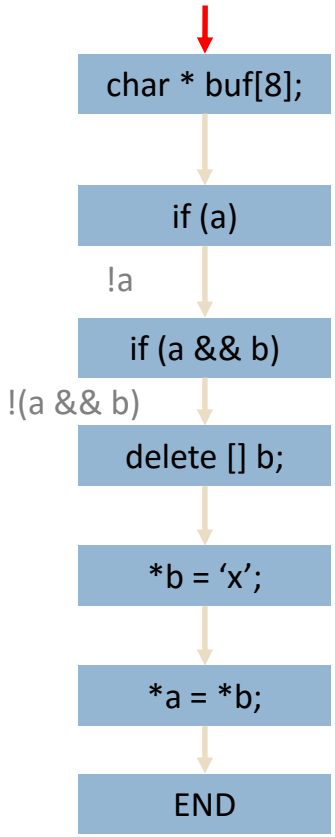
Path Traversal



Conceptually Analyze each path through control graph separately

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

Apply Checking



See how three checkers are run for this path

Checker

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

Run Checker

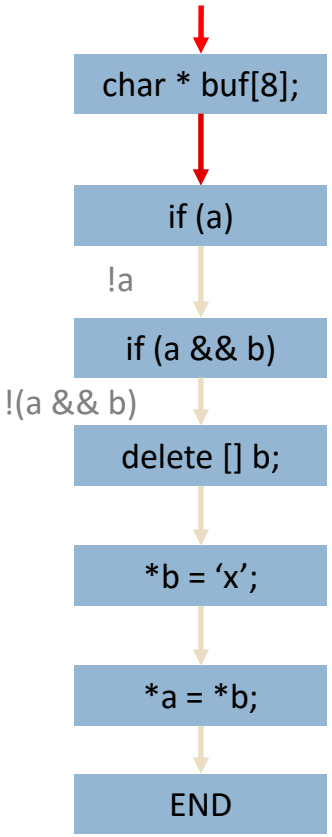
- Assign initial state to each program var
- State at program point depends on state at previous point, program actions
- Emit error if error state reached

Null pointers

Use after free

Array overrun

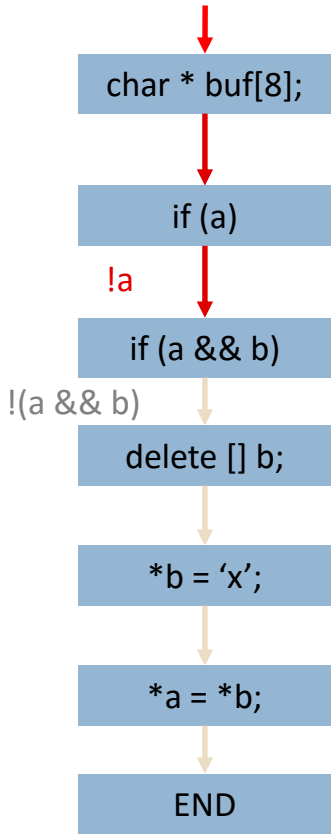
Apply Checking



Null pointers Use after free Array overrun

“buf is 8 bytes”

Apply Checking



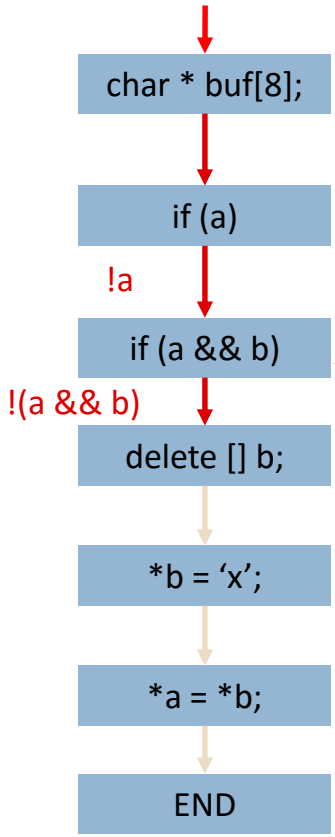
Null pointers Use after free Array overrun

"buf is 8 bytes"

"a is null"

Apply Checking

Null pointers Use after free Array overrun

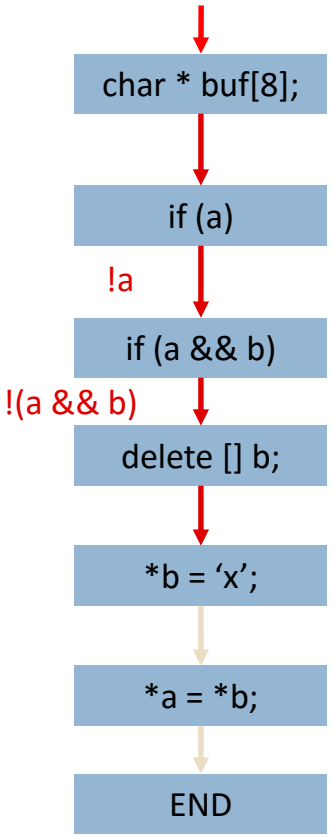


"buf is 8 bytes"

"a is null"

Already knew a was null

Apply Checking



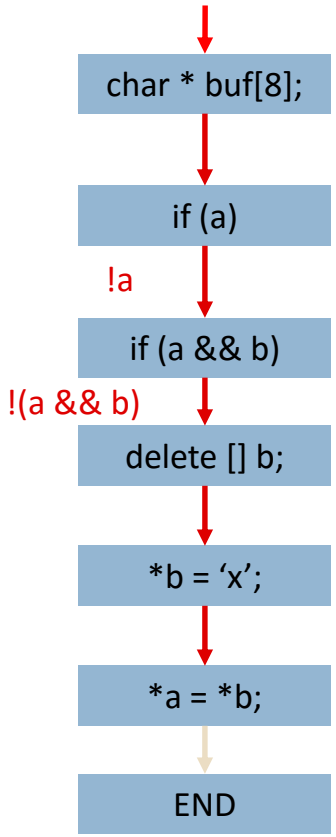
Null pointers Use after free Array overrun

“buf is 8 bytes”

“a is null”

“b is deleted”

Apply Checking



Null pointers Use after free Array overrun

"buf is 8 bytes"

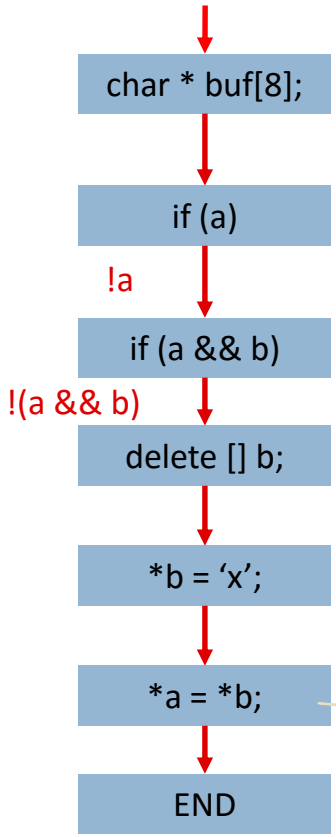
"a is null"

"b is deleted"

"b dereferenced!"

Apply Checking

Null pointers Use after free Array overrun



"buf is 8 bytes"

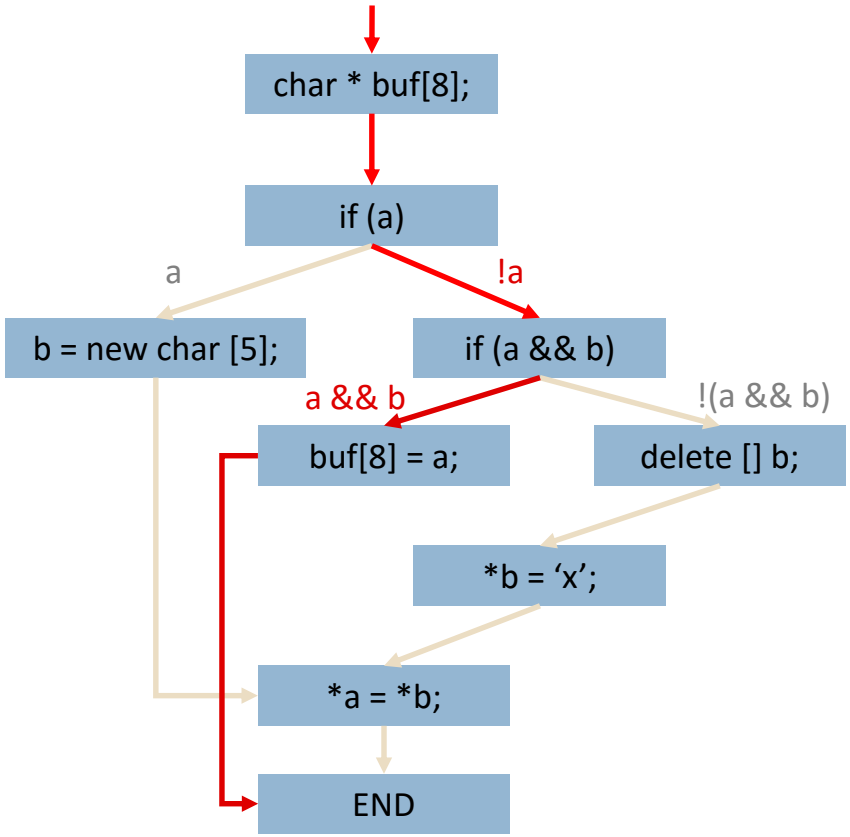
"a is null"

"b is deleted"

"b dereferenced!"

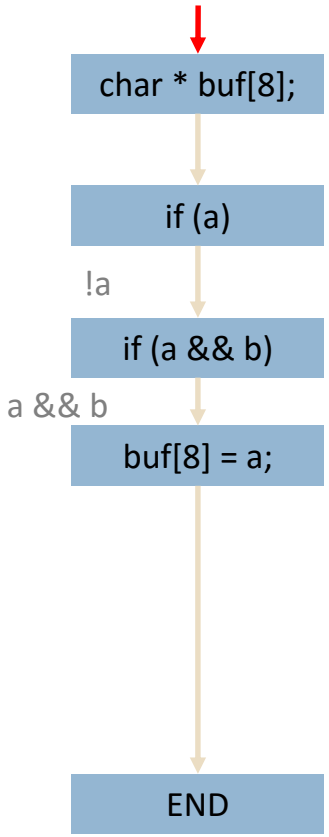
No more errors reported for b

A False Path

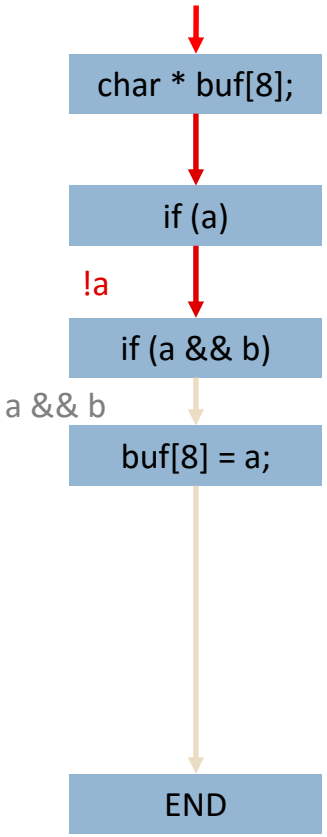


False Path Pruning

Integer Range Disequality Branch



False Path Pruning

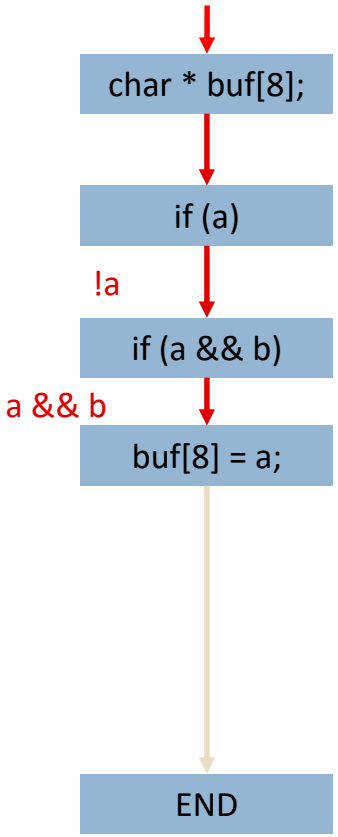


Integer Range | Disequality | Branch

"a in [0,0]"

"a == 0 is true"

False Path Pruning



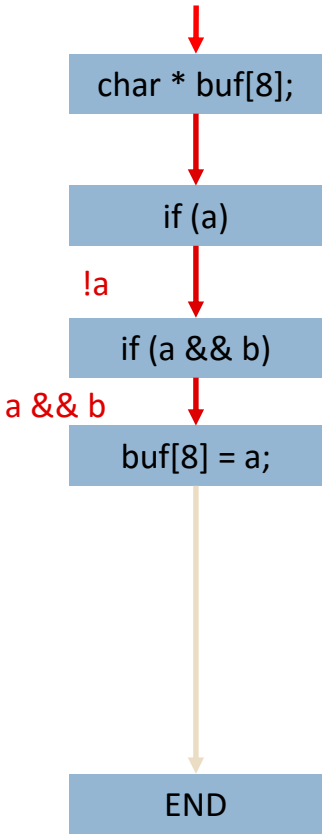
Integer Range | Disequality | Branch

"a in [0,0]"

"a == 0 is true"

"a != 0"

False Path Pruning



Integer Range Disequality Branch

Impossible

“a in [0,0]”

“a != 0”

“a == 0 is true”

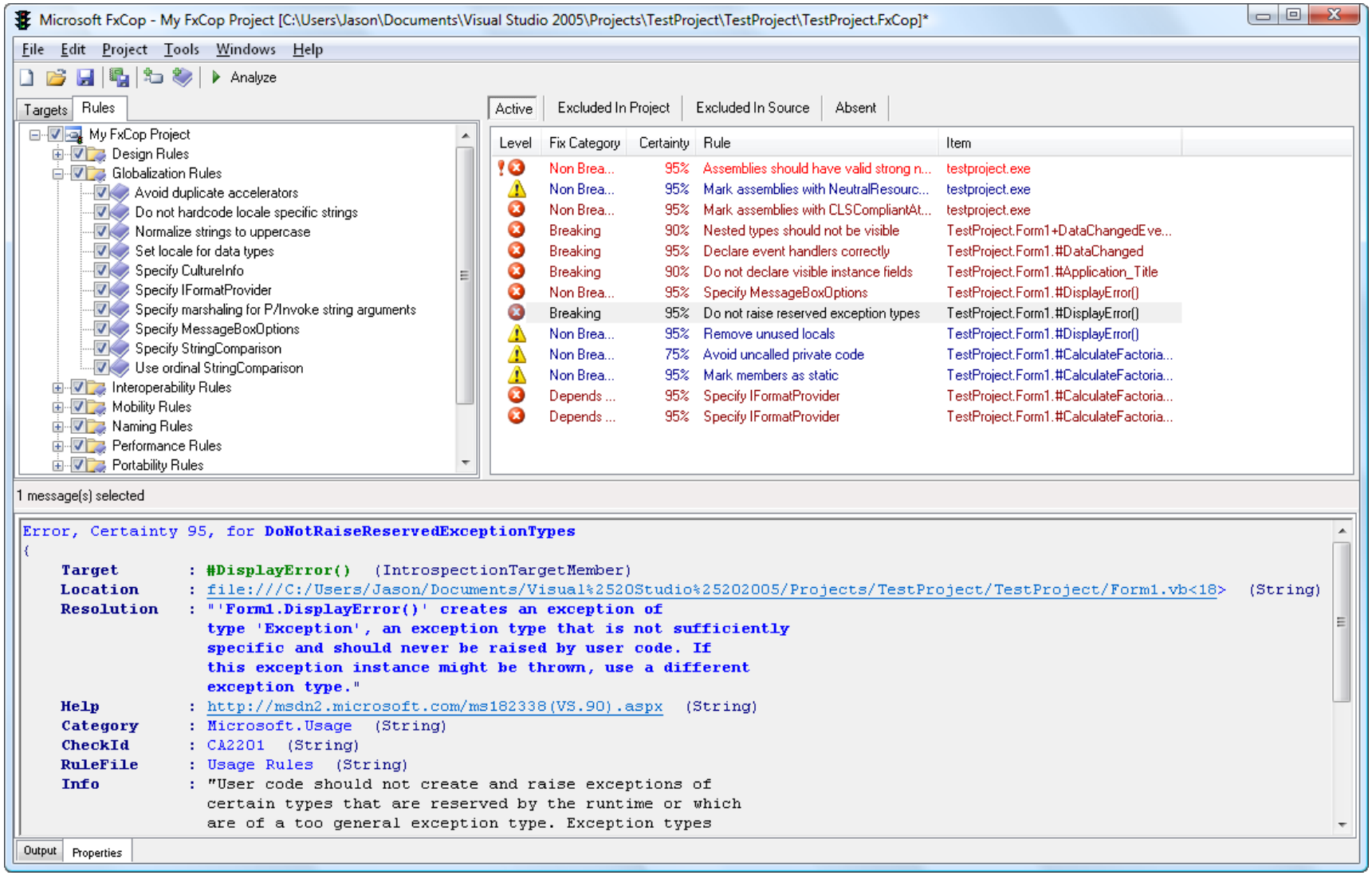
Results for BSD and Linux

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**

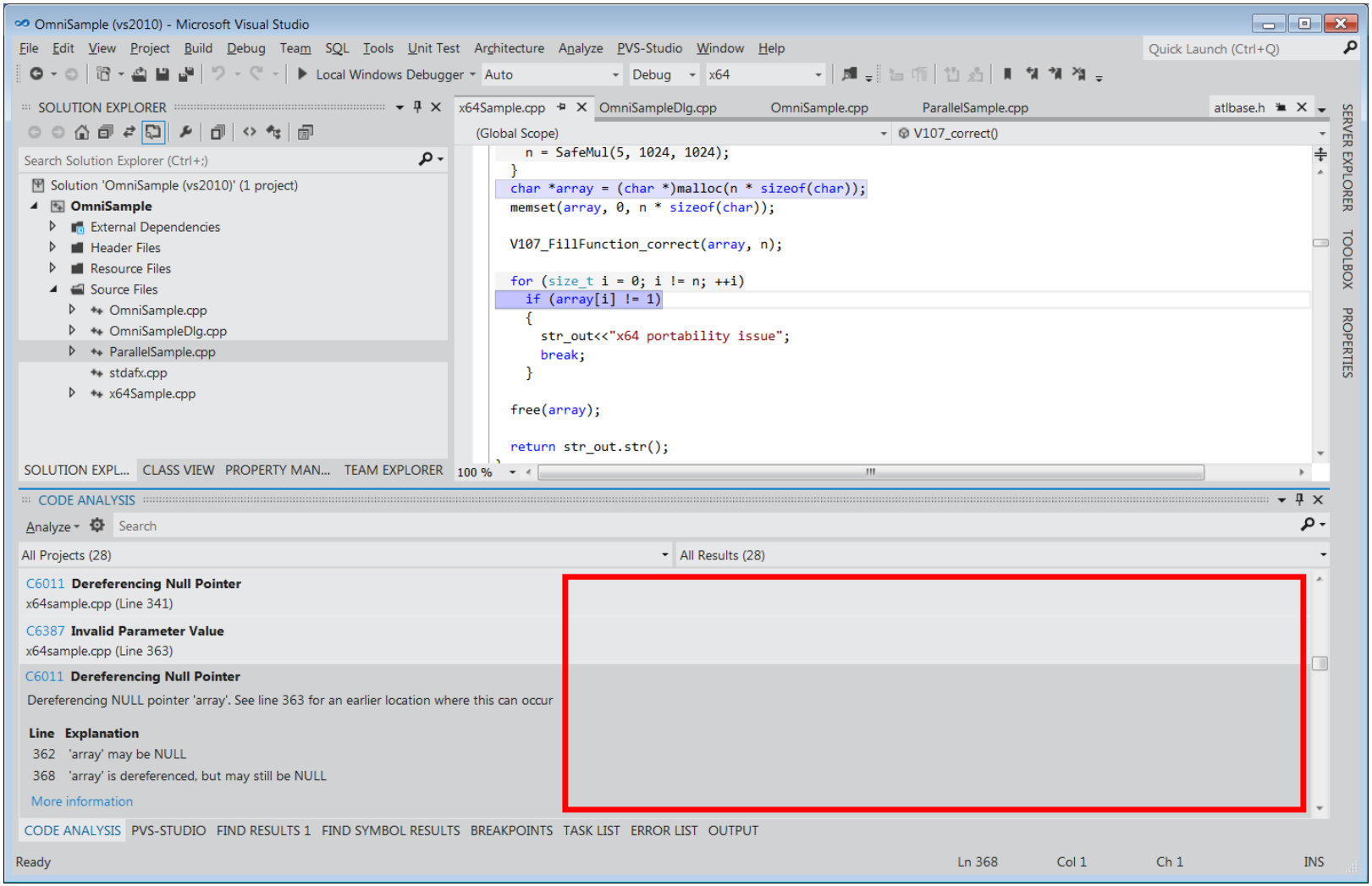
From CPyChecker


```
3 static PyObject* leaky(PyObject* self, PyObject* args) {
4     PyObject *leaked = PyString_FromString("leak!");
5     Py_XINCREF(leaked);
          when PyString_FromString() succeeds
          PyStringObject allocated at:    PyObject *leaked = PyString_FromString("leak!");
          ob_refcnt is now refs: 1 + N where N >= 0
6     return leaked;
          taking False path
          ob_refcnt is now refs: 2 + N where N >= 0
7 }
          returning
          ob_refcnt of return value is 1 too high
          was expecting final ob_refcnt to be N + 1 (for some unknown N)
          due to object being referenced by: return value
          but final ob_refcnt is N + 2
```

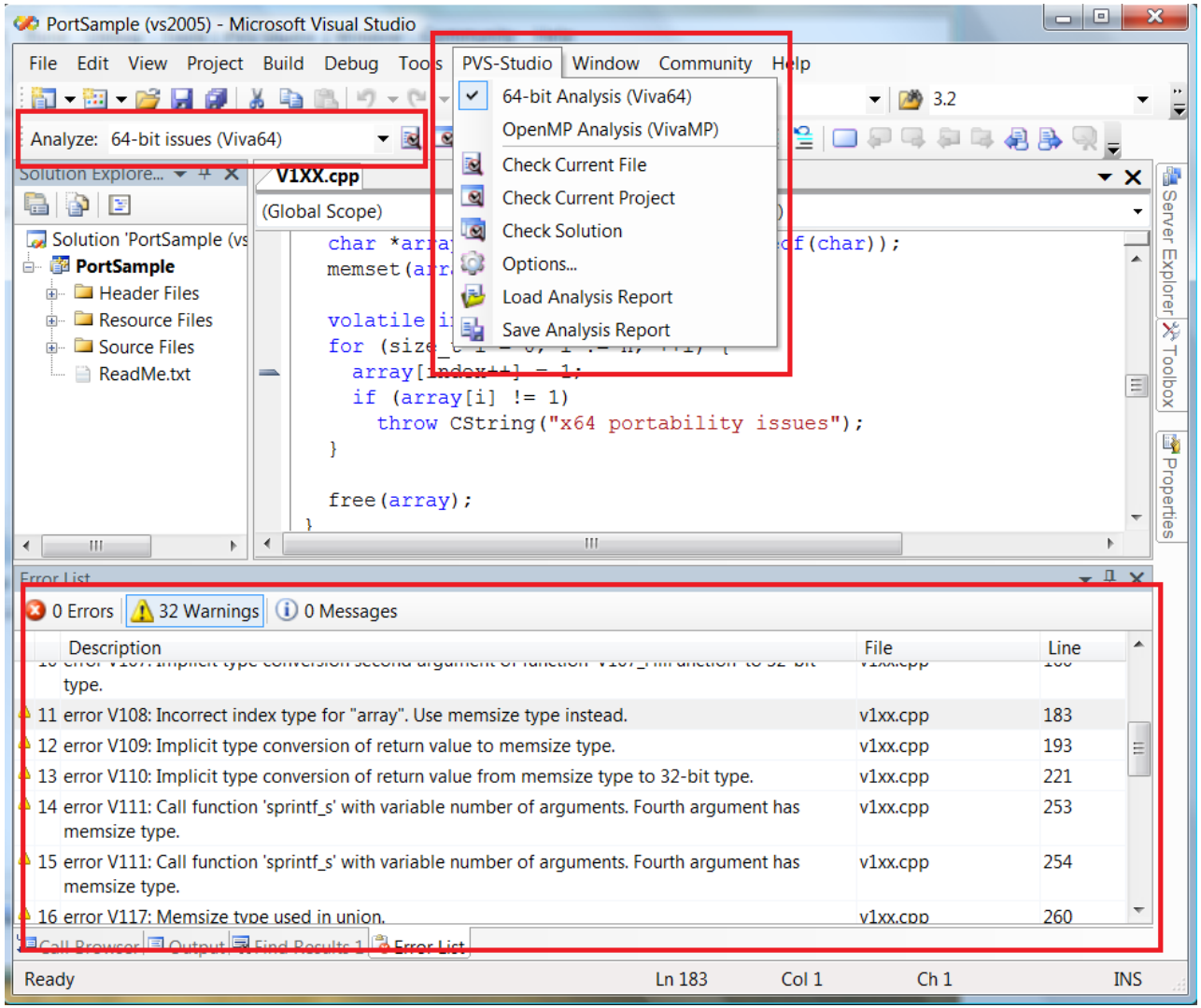
From FxCop


Microsoft FxCop - My FxCop Project [C:\Users\Jason\Documents\Visual Studio 2005\Projects\TestProject\TestProject\TestProject.FxCop]*
File Edit Project Tools Windows Help
Analyze
Targets Rules
My FxCop Project
Design Rules
Globalization Rules
Avoid duplicate accelerators
Do not hardcode locale specific strings
Normalize strings to uppercase
Set locale for data types
Specify CultureInfo
Specify IFormatProvider
Specify marshaling for P/Invoke string arguments
Specify MessageBoxOptions
Specify StringComparison
Use ordinal StringComparison
Interoperability Rules
Mobility Rules
Naming Rules
Performance Rules
Portability Rules
Active Excluded In Project Excluded In Source Absent
Level Fix Category Certainty Rule Item
Non Brea... 95% Assemblies should have valid strong n... testproject.exe
Non Brea... 95% Mark assemblies with NeutralResourc... testproject.exe
Non Brea... 95% Mark assemblies with CLSCompliantAt... testproject.exe
Breaking 90% Nested types should not be visible TestProject.Form1+DataChangedEve...
Breaking 95% Declare event handlers correctly TestProject.Form1.#DataChanged
Breaking 90% Do not declare visible instance fields TestProject.Form1.#Application_Title
Non Brea... 95% Specify MessageBoxOptions TestProject.Form1.#DisplayError()
Breaking 95% Do not raise reserved exception types TestProject.Form1.#DisplayError()
Non Brea... 95% Remove unused locals TestProject.Form1.#DisplayError()
Non Brea... 75% Avoid uncalled private code TestProject.Form1.#CalculateFactoria...
Non Brea... 95% Mark members as static TestProject.Form1.#CalculateFactoria...
Depends ... 95% Specify IFormatProvider TestProject.Form1.#CalculateFactoria...
Depends ... 95% Specify IFormatProvider TestProject.Form1.#CalculateFactoria...
1 message(s) selected
Error, Certainty 95, for DoNotRaiseReservedExceptionTypes
{
Target : #DisplayError() (IntrospectionTargetMember)
Location : file:///C:/Users/Jason/Documents/Visual%2520Studio%25202005/Projects/TestProject/TestProject/Form1.vb<18> (String)
Resolution : "'Form1.DisplayError()' creates an exception of type 'Exception', an exception type that is not sufficiently specific and should never be raised by user code. If this exception instance might be thrown, use a different exception type."
Help : http://msdn2.microsoft.com/ms182338(VS.90).aspx (String)
Category : Microsoft.Usage (String)
CheckId : CA2201 (String)
RuleFile : Usage Rules (String)
Info : "User code should not create and raise exceptions of certain types that are reserved by the runtime or which are of a too general exception type. Exception types
Output Properties

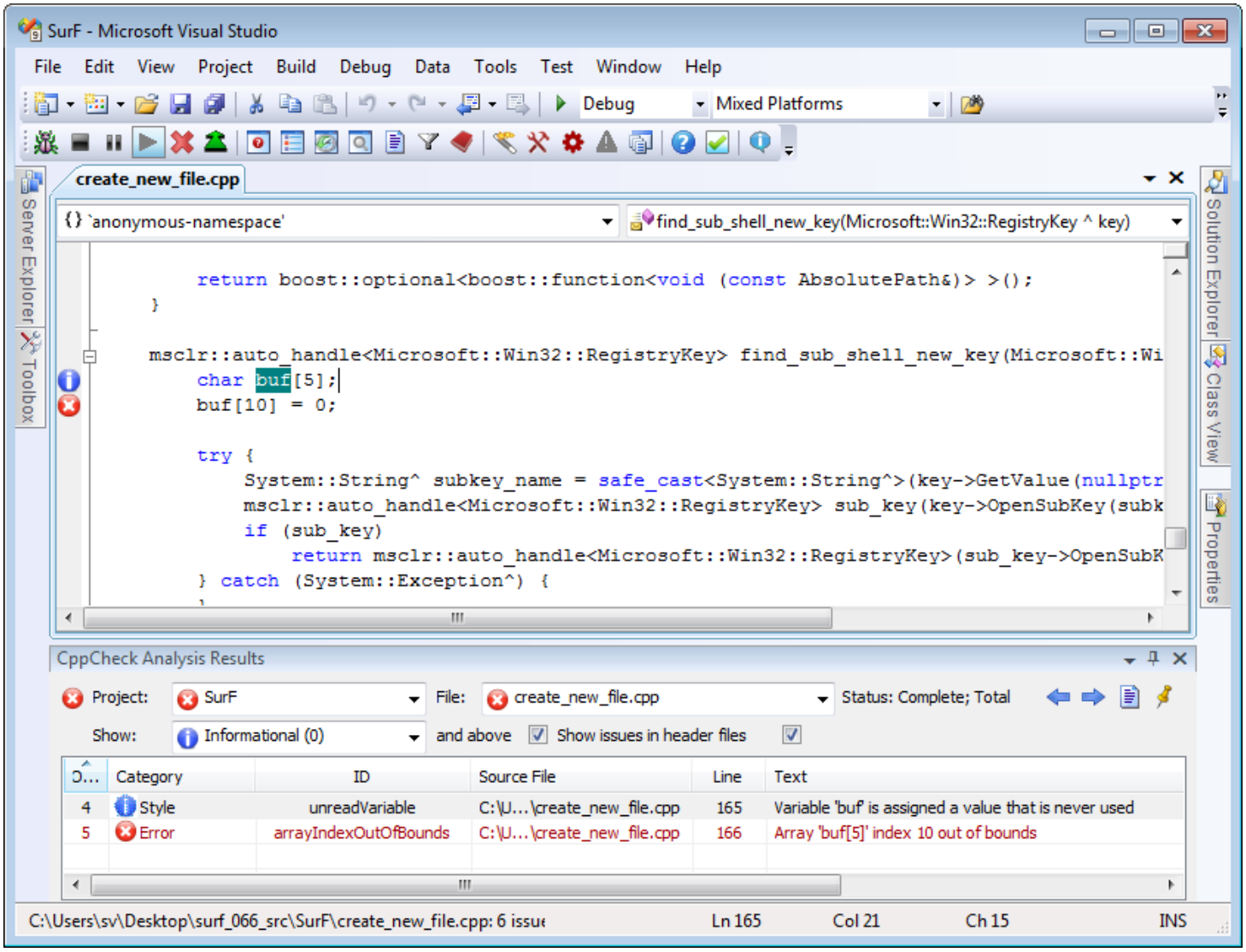
Visual Studio



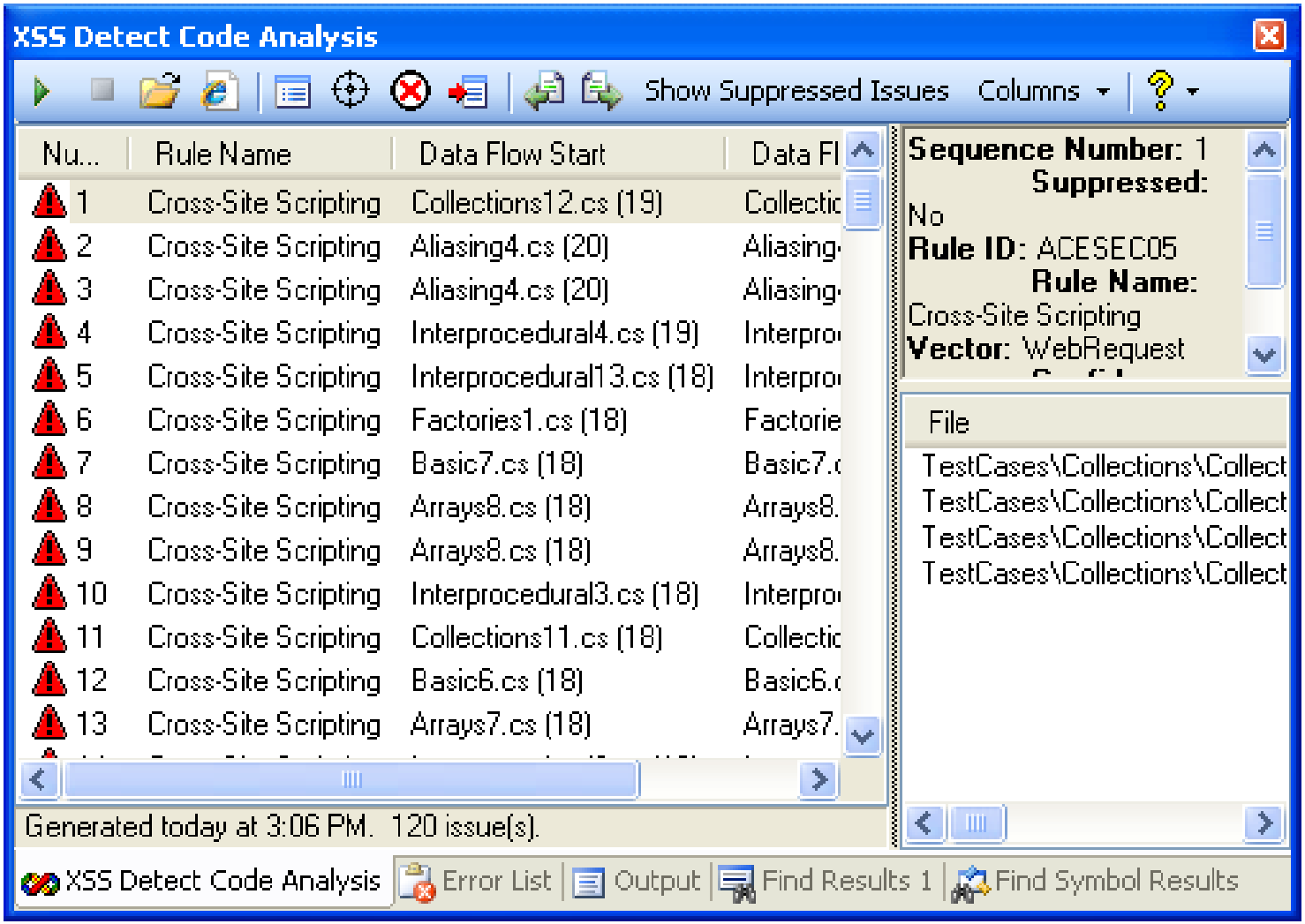
From PVS-Studio



From Visual Lint



XSS Detect


XSS Detect Code Analysis
Show Suppressed Issues
Columns
Nu...
Rule Name
Data Flow Start
Data Fl
1 Cross-Site Scripting Collections12.cs (19)
2 Cross-Site Scripting Aliasing4.cs (20)
3 Cross-Site Scripting Aliasing4.cs (20)
4 Cross-Site Scripting Interprocedural4.cs (19)
5 Cross-Site Scripting Interprocedural13.cs (18)
6 Cross-Site Scripting Factories1.cs (18)
7 Cross-Site Scripting Basic7.cs (18)
8 Cross-Site Scripting Arrays8.cs (18)
9 Cross-Site Scripting Arrays8.cs (18)
10 Cross-Site Scripting Interprocedural3.cs (18)
11 Cross-Site Scripting Collections11.cs (18)
12 Cross-Site Scripting Basic6.cs (18)
13 Cross-Site Scripting Arrays7.cs (18)
Sequence Number: 1
Suppressed:
No
Rule ID: ACESEC05
Rule Name:
Cross-Site Scripting
Vector: WebRequest
File
TestCases\Collections\Collect
TestCases\Collections\Collect
TestCases\Collections\Collect
TestCases\Collections\Collect
Generated today at 3:06 PM. 120 issue(s).
XSS Detect Code Analysis
Error List
Output
Find Results 1
Find Symbol Results

Approaches to Finding Security Bugs

- Static Analysis
- Runtime Monitoring
- **Black-box Testing**

Fuzzing Basics

- A form of vulnerability analysis and testing
- Many slightly anomalous test cases are input into the target application
- Application is monitored for any sign of error

Example

- Standard HTTP GET request
  - `GET /index.html HTTP/1.1`
- Anomalous requests
  - `AAAAAA...AAAA /index.html HTTP/1.1`
  - `GET ///////index.html HTTP/1.1`
  - `GET %n%n%n%n%n%n.html HTTP/1.1`
  - `GET /AAAAAAAAAAAAA.html HTTP/1.1`
  - `GET /index.html HTTTTTTTTTTTTTP/1.1`
  - `GET /index.html HTTP/1.1.1.1.1.1.1.1.1`
  - etc…

Early Successes (1989 Fuzz Project)

Utility	VAX (v)	Sun (s)	HP (h)	i386 (x)	AIX 1.1 (a)	Sequent (d)
adb	●○	●	●	○	–	–
as	●			●	●	●
awk						
bc				●○		
bib			–	–	–	–
calendar				–		
cat						
cb	●		●	●	○	●
cc						
/lib/ccom				–	–	●
checkeq				–		
checknr				–	–	
col	●○	●	●	●○	●	●
colcrt				–	–	
colrm				–	–	
comm						
compress						
/lib/cpp						
csh	●○					
dbx		*	–	–		

*● = utility crashed, ○ = utility hung, * = crashed on SunOS 3.2 but not on SunOS 4.0, ⊕ = crashed only on SunOS 4.0, not 3.2. – = utility unavailable on that system. ! = utility caused the operating system to crash.*

iPhone Security Flaw: July 2007

Shortly after the iPhone was released, a group of security researchers at Independent Security Evaluators decided to investigate how hard it would be for a **remote adversary** to compromise the private information stored on the device

acker's Life: Charlie Miller Keeps the
ı World On Its Toes

lacklisted him. Twitter hired him.

nd Early December 28, 2012 1:53 PM

By Rosalind Early

a green military jacket and cap, Charlie Miller explains to a group of hackers and cyber
ow he'd build a cyber army in North Korea. He boasts that his army could infiltrate mil
and power grids, interrupt cellphone service, and take over millions of computers arou
: the bargain-basement price of $49 million." In this presentation, delivered at the 2010
e in Las Vegas, he's included doctored photos of himself standing beside North Korea's
m Jong Il, with the premise that he's been kidnapped.

Success

- Within two weeks of part time work, we had successfully
  - discovered a vulnerability
  - developed a tool chain for working with the iPhone's architecture
  - created a proof-of-concept exploit capable of delivering files from the user's iPhone to a remote attacker
- Notified Apple of the vulnerability and proposed a patch.
- Apple subsequently resolved the issue.
- Released an advisory

CVE-2007-3944 Issued and Patched

WebKit

CVE-ID: CVE-2007-3944

Available for: Mac OS X v10.3.9, Mac OS X Server v10.3.9, Mac OS X v10.4.10, Mac OS X Server v10.4.10

Impact: Viewing a maliciously crafted web page may lead to arbitrary code execution

Description: Description: Heap buffer overflows exist in the Perl Compatible Regular Expressions (PCRE) library used by the JavaScript engine in Safari. By enticing a user to visit a maliciously crafted web page, an attacker may trigger the issue, which may lead to arbitrary code execution. This update addresses the issue by performing additional validation of JavaScript regular expressions. Credit to Charlie Miller and Jake Honoroff of Independent Security Evaluators for reporting these issues.

iPhone Attack

- iPhone Safari downloads malicious web page
  - Arbitrary code is run with administrative privileges
  - Can read SMS log, address book, call history, etc.
  - Can transmit collected data to attacker
  - Can perform physical actions on the phone
    - system sound and vibrate the phone for a second
    - could dial phone numbers, send text messages, or record audio (as a bugging device)

How Was This Discovered?

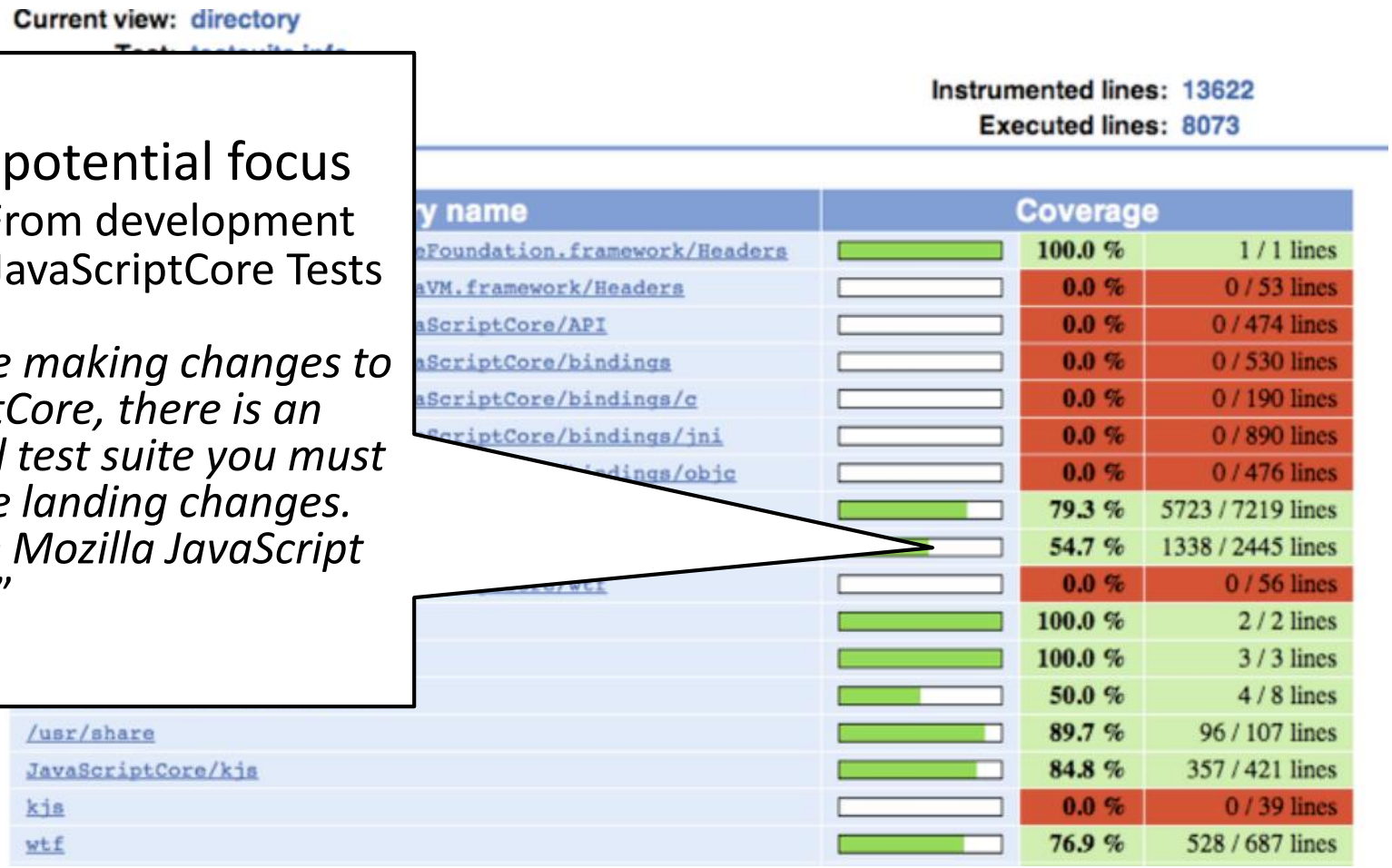
- WebKit is open source
  - “WebKit is an open source web browser engine. WebKit is also the name of the Mac OS X system framework version of the engine that's used by Safari, Dashboard, Mail, and many other OS X applications.”

- So we know what they use for code testing
  - Use code coverage to see which portions of code is not well tested
  - Tools gcov, icov, etc., measure test coverage

Collect Coverage for the Test Suite

Identify potential focus points. From development site: the JavaScriptCore Tests

"If you are making changes to JavaScriptCore, there is an additional test suite you must run before landing changes. This is the Mozilla JavaScript test suite."



Current view: directory

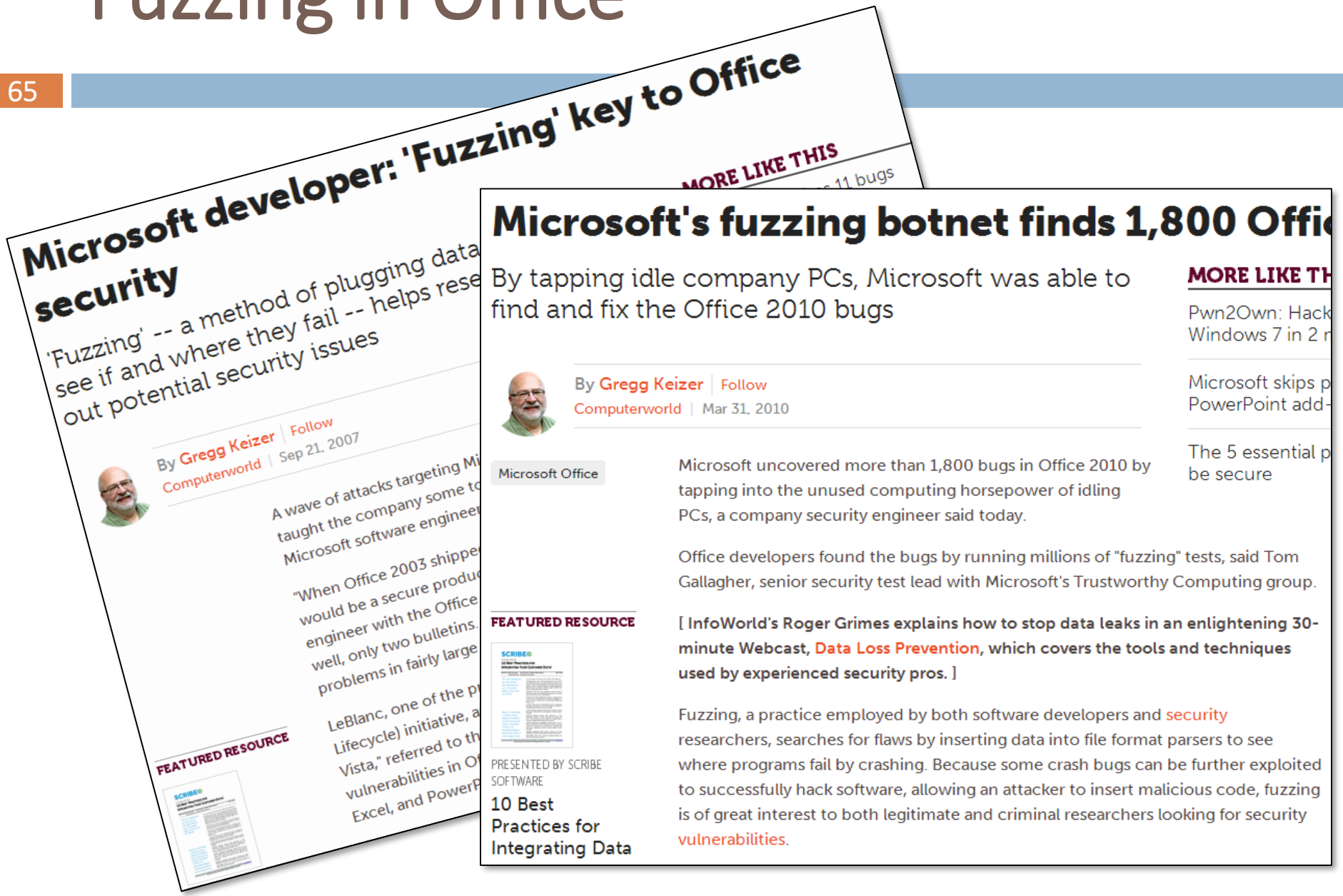
Instrumented lines: 13622
Executed lines: 8073

y name	Coverage		
eFoundation.framework/Headers		100.0 %	1 / 1 lines
aVM.framework/Headers		0.0 %	0 / 53 lines
aScriptCore/API		0.0 %	0 / 474 lines
aScriptCore/bindings		0.0 %	0 / 530 lines
aScriptCore/bindings/c		0.0 %	0 / 190 lines
ScriptCore/bindings/jni		0.0 %	0 / 890 lines
dings/objc		0.0 %	0 / 476 lines
		79.3 %	5723 / 7219 lines
		54.7 %	1338 / 2445 lines
/wtf		0.0 %	0 / 56 lines
		100.0 %	2 / 2 lines
		100.0 %	3 / 3 lines
		50.0 %	4 / 8 lines
/usr/share		89.7 %	96 / 107 lines
JavaScriptCore/kjs		84.8 %	357 / 421 lines
kjs		0.0 %	0 / 39 lines
wtf		76.9 %	528 / 687 lines

What To Focus?

- 59.3% of 13,622 lines in `JavaScriptCore` were covered
  - 79.3% of main engine covered
  - **54.7%** of Perl Compatible Regular Expression (PCRE) covered
- Next step: focus on PCRE
  - Wrote a PCRE fuzzer (20 lines of perl)
  - Ran it on standalone PCRE parser (pcredemo from PCRE library)
  - Started getting errors: *PCRE compilation failed at offset 6: internal error: code overflow*
- Evil regular expressions crash mobile Safari

Fuzzing in Office



Microsoft developer: 'Fuzzing' key to Office security

'Fuzzing' -- a method of plugging data see if and where they fail -- helps rese out potential security issues

By Gregg Keizer | Follow
Computerworld | Sep 21, 2007

A wave of attacks targeting Mi taught the company some to Microsoft software engineer

"When Office 2003 shippe would be a secure produ engineer with the Office well, only two bulletins. problems in fairly large

LeBlanc, one of the p Lifecycle) initiative, a Vista," referred to th vulnerabilities in Of Excel, and PowerP

FEATURED RESOURCE

MORE LIKE THIS

11 bugs

Microsoft's fuzzing botnet finds 1,800 Offi

By tapping idle company PCs, Microsoft was able to find and fix the Office 2010 bugs

By Gregg Keizer | Follow
Computerworld | Mar 31, 2010

Microsoft Office

Microsoft uncovered more than 1,800 bugs in Office 2010 by tapping into the unused computing horsepower of idling PCs, a company security engineer said today.

Office developers found the bugs by running millions of "fuzzing" tests, said Tom Gallagher, senior security test lead with Microsoft's Trustworthy Computing group.

[InfoWorld's Roger Grimes explains how to stop data leaks in an enlightening 30-minute Webcast, Data Loss Prevention, which covers the tools and techniques used by experienced security pros.]

Fuzzing, a practice employed by both software developers and security researchers, searches for flaws by inserting data into file format parsers to see where programs fail by crashing. Because some crash bugs can be further exploited to successfully hack software, allowing an attacker to insert malicious code, fuzzing is of great interest to both legitimate and criminal researchers looking for security vulnerabilities.

MORE LIKE TH

Pwn2Own: Hack Windows 7 in 2 r

Microsoft skips p PowerPoint add-

The 5 essential p be secure

FEATURED RESOURCE

PRESENTED BY SCRIBE SOFTWARE

10 Best Practices for Integrating Data

Mutation Based Fuzzing

- Little or no knowledge of the structure of the inputs is assumed
- Input anomalies are added to existing valid inputs
- Input anomalies may be completely random or follow some heuristics
- Requires little to no set up time
- Success dependent on the inputs being modified
- May help to get to parts of the code protected by complex conditionals
- May fail for protocols with checksums, those which depend on challenge response, etc.
- Examples:
  - ZZUF, very successful at finding bugs in many real-world programs, http://sam.zoy.org/zzuf/
  - Taof, GPF, ProxyFuzz, FileFuzz, Filep, etc.

Example: Fuzzing a PDF Viewer

- Google for .pdf (about 1 billion results)
- Crawl pages to build a corpus
- Use fuzzing tool (or script to)
  - 1. Grab a file
  - 2. Mutate that file
  - 3. Feed it to the program
  - 4. Record if it crashed (and input that crashed it)

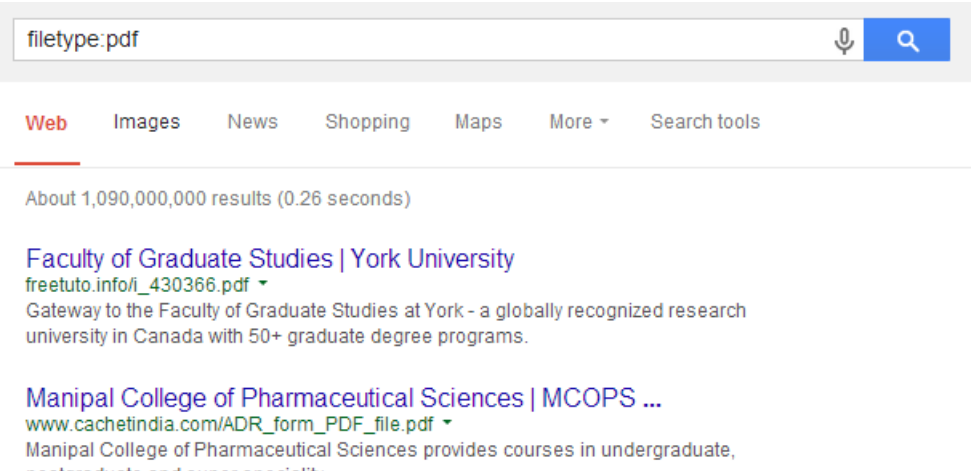


Image Format Fuzzing?

If you are reading this article on a computer, then the picture shown in Figure 1 is displayed on your screen after a jpg parser (typically part of your operating system) has read the image data, decoded it, created new data structures with the decoded data, and passed those to the graphics card in your computer. If the code implementing that jpg parser contains a bug such as a buffer overflow that can be triggered by a corrupted jpg image, then the execution of this jpg parser on your computer could potentially be hijacked to execute some other code, possibly malicious and hidden in the jpg data itself.

fuzzdb: Attack and Discovery Pattern Database for Application Fuzz Testing

A
TRUE
FALSE
0
00
1
-1
1.0
-1.0
2
-2
-20
65536
268435455
-268435455
2147483647
0xffffffff
NULL
null
\0
\00
< script > < / script>
%0a
%00
+%00
\0
\0\0

dir%00|
|dir
|dir|
|/bin/ls -al
?x=
?x="
?x=|
?x=>
/boot.ini
ABCD|%8.8x|%8.8x|%8.
8x|%8.8x|%8.8x|%8.8x|
%8.8x|%8.8x|%8.8x|%8.
8x|
../../boot.ini
/../../../../../../../../%2A
%25%5c..%25%5c..%25%
5c..%25%5c..%25%5c..%
25%5c..%25%5c..%25%5
c..%25%5c..%25%5c..%2
5%5c..%25%5c..%
25%5c..%2
5%5c..%00
%25%5c..%25%5c..%25%
5c..%25%5c..%25%5c..%
25%5c..%25%5c..%25%5
c..%25%5c..%25%5c..%2
5%5c..%25%5c..%

03C
&#x0003C
&#x00003C
&#x000003C
<
<
<
<
<
<
&#X3C
&#X03C
&#X003C
&#X0003C
&#X00003C
&#X000003C
<
<
<
<
<
<
\x3c

\x3C
\u003c
\u003C
something%00html
'
/'
\'
^'
@'
{'}
[']
*'
#'
">xxx<P>yyy
"><script>"
<script>alert("XSS")</scri
pt>
uname -n -s
whoami
pwd
last
cat /etc/passwd
ls -la /tmp
ls -la /home
ping -i 30 127.0.0.1
ping 127.0.0.1
ping -n 30

Grammar-based Based Fuzzing

- Test cases are generated from some description of the format: RFC, documentation, grammar, etc.
- Knowledge of format or protocol should give better results than random fuzzing
- Can take significant time to set up

```
from binascii import *
from struct import *

def insert_questions (sess, node, edge, sock):
        node.names['Questions'].value = 1+node.names['queries
        node.names['Authority'].value = 1+node.names['auth_na

s_initialize("query")
s_word(0, name="TransactionID")
s_word(0, name="Flags")
s_word(1, name="Questions", endian='>')
s_word(0, name="Answer", endian='>')
s_word(1, name="Authority", endian='>')
s_word(0, name="Additional", endian='>')

######### Queries ################
if s_block_start("query"):
        if s_block_start("name_chunk"):
                s_size("string", length=1)
                if s_block_start("string"):
                        s_string("A"*10)
                s_block_end()
        s_block_end()
        s_repeat("name_chunk", min_reps=2, max_reps=4, step=1

        s_group("end", values=["\x00", "\xc0\xb0"])   # very
        s_word(0xc, name="Type", endian='>')
        s_word(0x8001, name="Class", endian='>')
s_block_end()
s_repeat("query", 0, 1000, 40, name="queries")
```

Grammar-based: SPIKE


```
s_string("POST /testme.php
HTTP/1.1rn");
s_string("Host:
testserver.example.comrn");
s_string("Content-Length: ");
s_blocksize_string("block1",
5);
s_string("rnConnection:
closernrn");
s_block_start("block1");
s_string("inputvar=");
s_string_variable("inputval")
;
s_block_end("block1");
```

```
POST /testme.php HTTP/1.1
Host: testserver.example.com
Content-Length: [size_of_data]
Connection: close
inputvar=[fuzz_string]
```

s_string_variable(“string”); // inserts a fuzzed string into your “SPIKE”.

The string “string” will be used for the first iteration of this variable, as well as for any SPIKES where other s_string_variables are being iterated

The Problems With Fuzzing

- Mutation based fuzzers can generate a huge number of test cases... When has the fuzzer run long enough?
- Generation based fuzzers generate lots of test cases, too. What happens when they're all run and no bugs are found?
- How do you monitor the target application such that you know when something "bad" has happened?

More Issues with Fuzzing

- What happens when you find too many bugs?
- Or every anomalous test case triggers the same (boring) bug?
- Given a crash, how do you find the actual vulnerability
- After fuzzing, how do you know what changes to make to improve your fuzzer?
- When do you stop fuzzing an application?

Example: PDF

- Have a PDF file with 248,000 bytes
  - There is one byte that, if changed to **particular values**, causes a crash
  - This byte is 94% of the way through the file
- Any single random mutation to the file has a probability of .00000392 of finding the crash
- On average, need 127,512 test cases to find it
- At 2 seconds a test case, that's just under 3 days

Example: 3g2 Video Files

- Changing a byte in the file to 0xff crashes QuickTime Player 42% of the time
- All these crashes seem to be from the same bug
- There may be other bugs “hidden” by this bug

Apple QuickTime 3g2 'mp4v' atom size Remote Code Execution Vulnerability

ZDI-11-277: August 31st, 2011

CVE ID

CVE-2011-0258

CVSS Score

7.5, (AV:N/AC:L/Au:N/C:P/I:P/A:P)

Affected Vendors

Apple

Affected Products

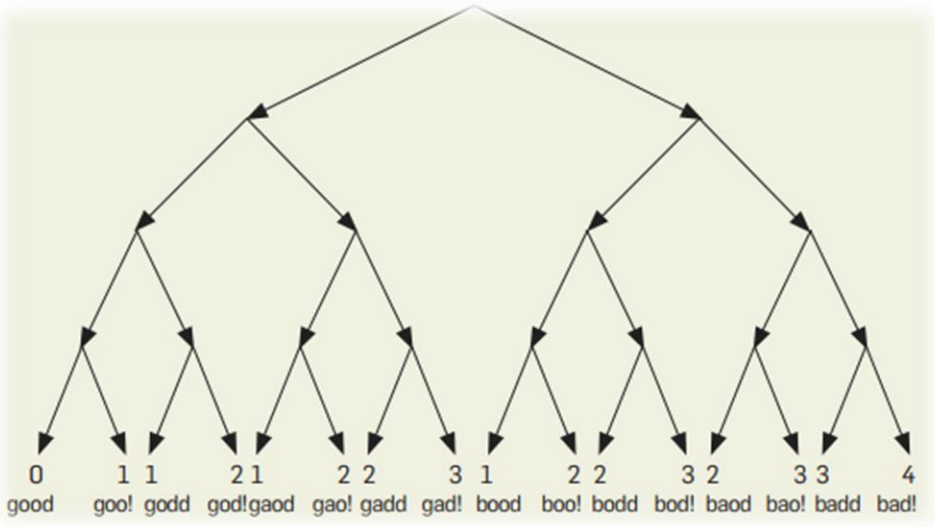
Quicktime

Types of Code Coverage

- Line/block coverage
  - Measures how many lines of source code have been executed.
- Branch coverage
  - Measures how many branches in code have been taken (conditional jmps)
- Path coverage
  - Measures how many paths have been taken

Path Coverage Issues

- In general, a program with n "reachable" branches will require 2n test cases for branch coverage and 2^n test cases for path coverage!
- If you consider loops, there are an infinite number of paths
- Some paths are infeasible
- You can't satisfy both of these conditionals, i.e. there is only three paths through this code, not four




```
if(x>=0){
   x = 1;
}
if(x < 0) {
   x = -1;
}
```