LAVA: Large-scale Automated Vulnerability Addition

Tim Leek, Patrick Hulin, Ryan Whelan (MIT/LL),

Brendan Dolan-Gavitt (NYU),

Fredrick Ulrich, Andrea Mambretti, Wil Robertson, and Engin Kirda (Northeastern)

LAVA

May 22, 2016

LINCOLN LABORATORY
MASSACHUSETTS INSTITUTE OF TECHNOLOGY

This work is sponsored by the Assistant Secretary of Defense for Research and Engineering under Air Force Contract FA8721-05-C-0002. Opinions, interpretations, conclusions and recommendations are those of the author and are not necessarily endorsed by the United States Government.

The problem: vulnerability discovery

NEWS

Extremely severe bug leaves dizzying number of software and devices vulner
Since 2008, vulnerability has left apps and hardware open to remote hijacking.
by Dan Goodin - Feb 16, 2016 2:01pm EST

```
[root@sandbox-3]$ gcc -o client client.c
[root@sandbox-3]$
[root@sandbox-3]$ ./client
Segmentation fault (core dumped)
```

950 million Android phones can be hijack by malicious text messages
Booby-trapped MMS messages and websites exploit flaw in heart of Android.
by Dan Goodin - Jul 27, 2015 12:43pm EDT

Hacking Linked to China Exposes Millions of U.S. Workers
By DAVID E. SANGER and JULIE HIRSCHFELD DAVIS JUNE 4, 2015

WASHINGTON — The Obama administration on Thursday announced what appeared to be one of the largest breaches of federal employees' data, involving at least four million current and former government workers in an intrusion that officials said apparently originated in China.

ACADEMIA

An Empirical Study of the Reliability of UNIX Utilities
Barton P. Miller, bart@cs.wisc.edu
Lars Fredriksen, L.Fredriksen@att.com
Bryan So, so@cs.wisc.edu

1990

VTT PUBLICATIONS 448
A Functional Method for Assessing Protocol Implementation Security
Rauli Kaksonen
VTT Electronics

1995

KLEE: Unassisted and Automatic Generation of High-Coverage Tests for Complex Systems Programs
Cristian Cadar, Daniel Dunbar, Dawson Engler *
Stanford University

Abstract
We present a new symbolic execution tool, KLEE, capable of automatically generating tests that achieve high coverage on a diverse set of complex and environmentally-intensive programs.

2005

Driller: Augmenting Fuzzing Through Selective Symbolic Execution
Nick Stephens, John Grosen, Christopher Salls, Andrew Dutcher, Ruoyu Wang, Jacopo Corbetta, Yan Shoshitaishvili, Christopher Kruegel, Giovanni Vigna
UC Santa Barbara
{stephens,jmg,salls,dutcher,fish,jacopo,yans,chris,vigna}@cs.ucsb.edu

2016

INDUSTRY

klocwork — a Rogue Wave Company
∀Secure
VERACODE
coverity

DOES ANY OF THIS WORK?

Existing vulnerability corpora

National Vulnerability Database
nvd.nist.gov

CVE

SAMATE
NIST
National Institute of Standards and Technology

2004

Testing Static Analysis Tools using
Exploitable Buffer Overflows from Open Source Code

2005

Using a Diagnostic Corpus of C Programs to Eval…
Buffer Overflow Detection by Static Analysis Tools

Kendra Kratkiewicz
MIT Lincoln Laboratory
244 Wood Street
Lexington, MA 02420-9108
Phone: 781-981-2931
Email: KENDRA@LL.MIT.EDU

Richard Lippmann
MIT Lincoln Laboratory
244 Wood Street
Lexington, MA 02420-9108
Phone: 781-981-2711
Email: LIPPMANN@LL.MIT.EDU

ABSTRACT

A corpus of 291 small C-program test cases was developed to evaluate static and dynamic analysis tools designed to detect buffer overflows. The corpus was designed and labeled using a new, comprehensive buffer overflow taxonomy. It provides a benchmark to measure detection, false alarm, and confusion rates of tools, and also suggests areas for tool enhancement. Experiments with five tools demonstrate that some modern static analysis tools can accurately detect overflows in simple test cases but that others have serious limitations. For example, PolySpace demonstrated a superior detection rate, missing only one detection. Its performance could be enhanced if extremely long run times were reduced, and false alarms were eliminated for some C library functions. ARCHER performed well with no false alarms whatsoever. It could be enhanced by improving inter-procedural analysis and handling of C library functions. Splint

for a significant percentage of the software vulnerabilities published each year [17, 19], such as in NIST's ICAT Metabase [8], CERT advisories [1], Bugtraq [16], and other security forums. Buffer overflows have also been the basis for many damaging exploits, such as the Sapphire/Slammer [12] and Blaster [14] worms.

A buffer overflow vulnerability occurs when data can be written outside the memory allocated for a buffer, either past the end or before the beginning. Buffer overflows may occur on the stack, on the heap, in the data segment, or the BSS segment (the memory area a program uses for uninitialized global data), and may overwrite from one to many bytes of memory outside the buffer. Even a one-byte overflow can be enough to allow an exploit [9]. Buffer overflows have been described at length in many papers, including [19], and many descriptions of exploiting buffer overflows can be found online.

EXPLOIT DATABASE

metasploit Framework

ADOBE READER	\$5,000–\$30,000
MAC OSX	\$20,000–\$50,000
ANDROID	\$30,000–\$60,000
FLASH OR JAVA BROWSER PLUG-INS	\$40,000–\$100,000
MICROSOFT WORD	\$50,000–\$100,000
WINDOWS	\$60,000–\$120,000
FIREFOX OR SAFARI	\$60,000–\$150,000
CHROME OR INTERNET EXPLORER	\$80,000–\$200,000
IOS	\$100,000–\$250,000

Forbes, 2012

Vulnerability corpora sources

Source	Cost	Realism	Yield
Accident		High	Tiny
Search	$$$$	Med-High	Low
Injection	$$	Med	Low-Med
Synthesis	$	Low	High

LAVA

LAVA concept

- **Vulnerability corpus requirements**
  - ❑ Cheap and plentiful
  - ❑ Realistic
  - ❑ Triggering input
  - ❑ Manifest only for one or very few inputs
  - ❑ Security-critical effect

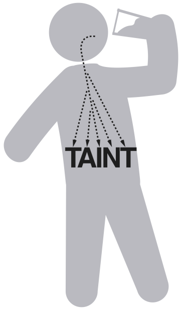
- **Caveats**
  - – Works only on source
  - – C programs
  - – Linux
  - – Buffer overflows

- **Large-scale Automated Vulnerability Addition**
  - – Uses static and dynamic analysis to find attacker-controlled data that can be used to introduce new code that creates a bug
  - – Change program and input at same time to insert bugs in known places
  - – Special sauce: new taint-based measures

LINCOLN LABORATORY
MASSACHUSETTS INSTITUTE OF TECHNOLOGY

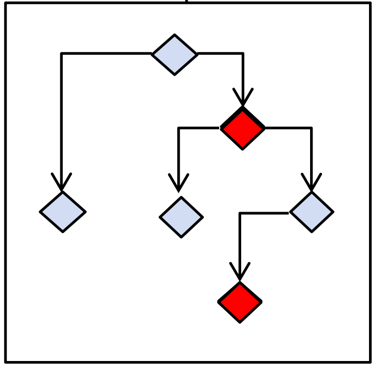
Dynamic taint analysis

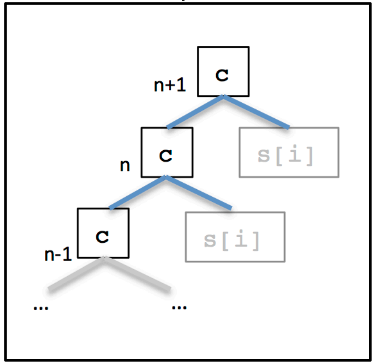
- **PANDA dynamic taint**
  - Whole system (all processes + kernel)
  - Works on binaries
  - Includes all library code
  - Oddball x86 instructions all analyzed including FPU and SSE
  - Many labels supported: Every byte in 10MB file
  - Labels combine into sets to represent computation
  - Fast (enough). 50-100x



https://github.com/panda-re

Taint-based measures





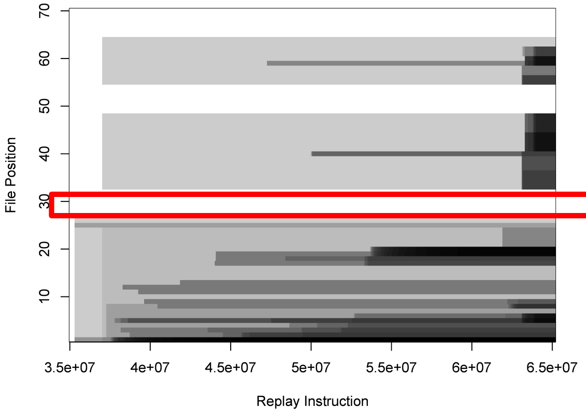
DEAD, UNCOMPLICATED, and AVAILABLE data (DUA)

Attacker-controlled data that can be used to create a vulnerability

Liveness:
Number of branches an input byte is used to decide.
How much effect upon control flow do specific input bytes have?

Taint compute number:
Depth of lval tree of computation.
How complicated a function of input bytes is an lval?

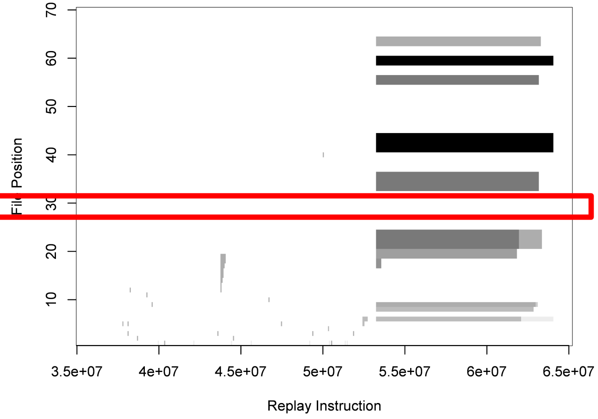
Taint-based measures



Liveness:
Number of branches an input byte is used to decide.
How much effect upon control flow do specific input bytes have?

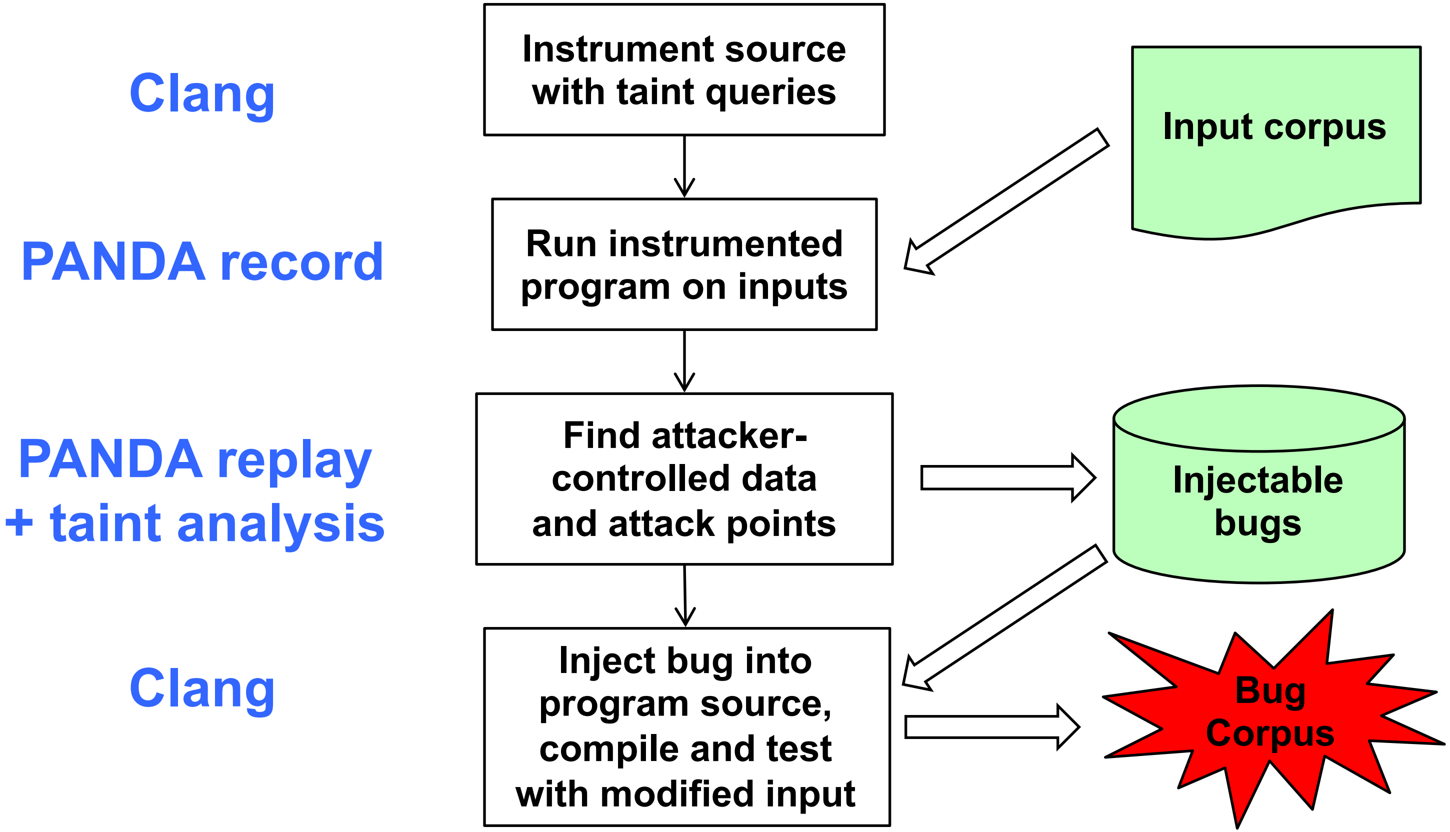
DEAD, UNCOMPLICATED, and AVAILABLE data (DUA)

Attacker-controlled data that can be used to create a vulnerability



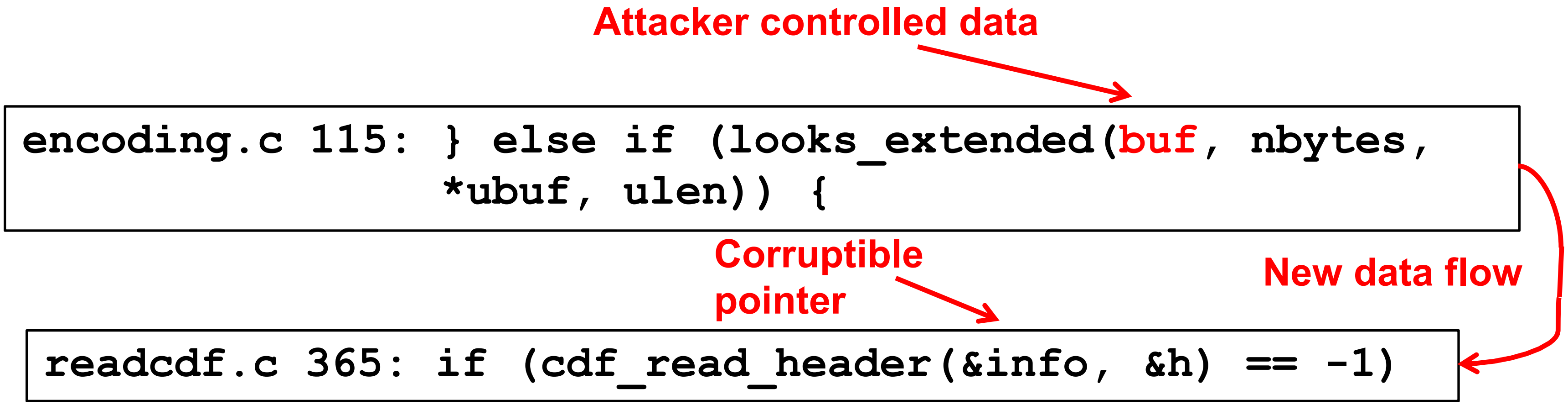
Taint compute number:
Depth of lval tree of computation.
How complicated a function of input bytes is an lval?

LAVA Taint-based bug injection



LAVA bug example

- PANDA taint analysis tells us that bytes 0-3 in the buffer `buf` at line 115 of `src/encoding.c` is attacker-controlled
- We also learn from PANDA that there is a pointer we can corrupt, ‘`&info`’, later in the execution, in `src/readelf.c`



LAVA bug example

- PANDA taint analysis tells us that bytes 0-3 in the buffer `buf` at line 115 of `src/encoding.c` is attacker-controlled
- We also learn from PANDA that there is a pointer we can corrupt, ‘`&info`’, later in the execution, in `src/readelf.c`

Attacker controlled data

```
encoding.c 115: } else if (looks_extended(buf, nbytes,
                   *ubuf, ulen)) {
```

Corruptible pointer

New data flow

```
readcdf.c 365: if (cdf_read_header(&info, &h) == -1)
```


LINCOLN LABORATORY
MASSACHUSETTS INSTITUTE OF TECHNOLOGY

LAVA bug example

```
// encoding.c:
} else if
  (({int rv =
        looks_extended(buf, nbytes, *ubuf, ulen);
      if (buf) {
        int lava = 0;
        lava |= ((unsigned char *)buf)[0];
        lava |= ((unsigned char *)buf)[1] << 8;
        lava |= ((unsigned char *)buf)[2] << 16;
        lava |= ((unsigned char *)buf)[3] << 24;
        lava_set(lava);
     }; rv; })) {
```

```
// readcdf.c:
if (cdf_read_header
     ((&info) + (lava_get()) *
       (0x6c617661 == (lava_get()) || 0x6176616c == (lava_get())),
        &h) == -1)
```

Vulnerability injection effectiveness

TABLE I
LAVA INJECTION RESULTS FOR OPEN SOURCE PROGRAMS OF VARIOUS SIZES

Name	Version	Num Src Files	Lines C code	N(DUA)	N(ATP)	Potential Bugs	Validated Bugs	Yield	Inj Time (sec)
file	5.22	19	10809	631	114	17518	774	38.7%	16
readelf	2.25	12	21052	3849	266	276367	1064	53.2 %	354
bash	4.3	143	98871	3832	604	447645	192	9.6%	153
tshark	1.8.2	1272	2186252	9853	1037	1240777	354	17.7%	542

Over 200K possible?

- **Four open source programs 10K -> 2M LOC**
- **2000 injection attempts per target (of over 1M)**
- **LAVA yield (validated injected bugs): 10->50%**
- **Over 2000 bugs injected**

Using LAVA to evaluate tools

- Created two corpora using LAVA
  - LAVA-1 programs containing individual bugs of varying difficulty
  - LAVA-M programs each with more than one bug
- Evaluated two open-source vulnerability discovery tools by ability to detect LAVA bugs
  - Fuzzer
  - Symbolic execution + SAT solving

TABLE IV
BUGS FOUND IN *LAVA-M* CORPUS BY TOOL TYPE

Tool Name	Total Bugs	Unique Bugs Found		
		FUZZER	SES	Combined
`uniq`	28	7	0	7
`base64`	44	7	9	14
`md5sum`	57	2	0	2
`who`	2136	0	18	18
Total	2265	16	27	41

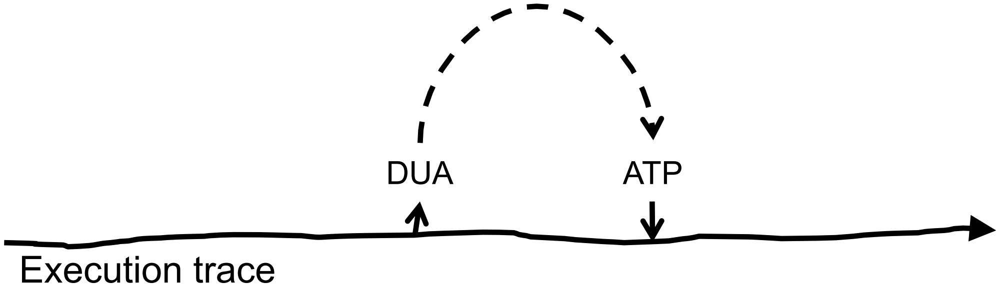
Detection < 2%

LAVA vulnerability realism

Realism is a concern. But hard to quantify

One possible measure is the fraction of the trace that is unaffected by LAVA yet must be analyzed correctly to discover the vulnerability

LAVA's bugs are inserted, generally quite far along in the trace. If anything we need some easier ones



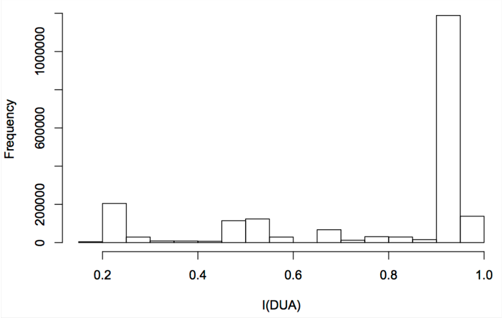


Fig. 8. Normalized DUA trace location

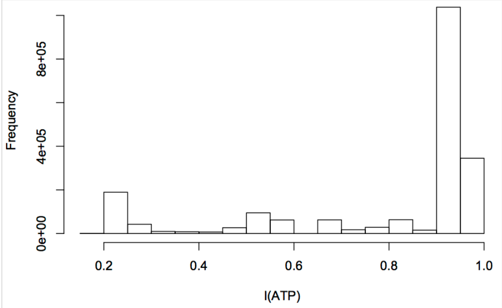


Fig. 9. Normalized ATP trace location

LINCOLN LABORATORY
MASSACHUSETTS INSTITUTE OF TECHNOLOGY

Summary and future directions

- Summary
  - Working system automates construction of large corpora for study and assessments
  - Novel taint-based measures are key: liveness and TCN
- Future directions
  - Continuous on-line competition to encourage self-eval
  - Use in security competitions like Capture the Flag to re-use and construct challenges on-the-fly
  - Assess and improve realism of LAVA bugs
  - More types of vulnerabilities
  - More interesting effects (exploitable ones)