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)

May 22, 2016

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

Extremely severe bug leaves dizzying number of software and devices vulnerable
Since 2008, vulnerability has left apps and hardware open to remote hijacking.

950 million Android phones can be hijacked by malicious text messages
Bobby-trapped MMS messages and websites exploit flaw in heart of Android.

Hacking Linked to China Exposes Millions of U.S. Workers
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.

NEWS

An Empirical Study of the Reliability of UNIX Utilities
Barbara P. Miller
Larry Friedman
Bruno Scarf

A Functional Method for Assessing Protocol Implementation Security
Paul Kasneren
VTT Electronics

KLEE: Unassisted and Automatic Generation of High-Coverage Tests for Complex Systems Programs
Christian Ceder, Daniel Berber, Daniel Engler * Stanford University

Driver: Augmenting Through Selective Symbolic Execution
Nick Stephens, John Gieson, Christopher Sulli, Andrew Dutcher, Roya Wang, Jacopo Corbet, Yan Shouleika, Christopher Kruegel, Giovanni Vigna
UC Santa Barbara


ACADEMIA

INDUSTRY

DOES ANY OF THIS WORK?

COVERAGE

VERACODE

VERACODE
Existing vulnerability corpora

Forbes, 2012
## Vulnerability corpora sources

<table>
<thead>
<tr>
<th>Source</th>
<th>Cost</th>
<th>Realism</th>
<th>Yield</th>
</tr>
</thead>
<tbody>
<tr>
<td>Accident</td>
<td>Free</td>
<td>High</td>
<td>Tiny</td>
</tr>
<tr>
<td>Search</td>
<td>$$$$</td>
<td>Med-High</td>
<td>Low</td>
</tr>
<tr>
<td>Injection</td>
<td>$$</td>
<td>Med</td>
<td>Low-Med</td>
</tr>
<tr>
<td>Synthesis</td>
<td>$</td>
<td>Low</td>
<td>High</td>
</tr>
</tbody>
</table>
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
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

**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?
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

Clang
- Instrument source with taint queries
  - Input corpus
- Run instrumented program on inputs
  - Find attacker-controlled data and attack points
  - Injectable bugs
- Inject bug into program source, compile and test with modified input
  - Bug Corpus

PANDA record
- PANDA replay + taint analysis
  - Clang
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

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

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

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

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

- Attacker controlled data
- Corruptible pointer
- New data flow
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

- 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

**TABLE I**

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

Over 200K possible?
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>
<thead>
<tr>
<th>Tool Name</th>
<th>Total Bugs</th>
<th>Unique Bugs Found</th>
<th>FUZZER</th>
<th>SES</th>
<th>Combined</th>
</tr>
</thead>
<tbody>
<tr>
<td>uniq</td>
<td>28</td>
<td></td>
<td>7</td>
<td>0</td>
<td>7</td>
</tr>
<tr>
<td>base64</td>
<td>44</td>
<td></td>
<td>7</td>
<td>9</td>
<td>14</td>
</tr>
<tr>
<td>md5sum</td>
<td>57</td>
<td></td>
<td>2</td>
<td>0</td>
<td>2</td>
</tr>
<tr>
<td>who</td>
<td>2136</td>
<td></td>
<td>0</td>
<td>18</td>
<td>18</td>
</tr>
<tr>
<td>Total</td>
<td>2265</td>
<td>16</td>
<td>27</td>
<td>41</td>
<td></td>
</tr>
</tbody>
</table>

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
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)