Industrial Application of Concolic Testing to Detect Crash Bugs - A Case Study on libexif

Yunho Kim, Moonzoo Kim, YoungJoo Kim, and Yoonkyu Jang
Provable SW Lab, KAIST, Samsung Electronics
South Korea

Content

- Motivation and project scope
- `libexif` case study
- Lessons learned and conclusion

Main Talk Summary

- Industry builds products based on OSS heavily
- Concolic testing is a good technique for **testing open source programs with modest effort**
  - We applied concolic testing to an open-source program `libexif` and detected 6 crash bugs in 4 man-week (reported 2 security bugs to CVE)

Motivation

- Effective SW code testing is expensive
  - Test oracle should be defined
    - Explicit high-level requirements are necessary
    - Target code knowledge is necessary to insert concrete low-level assert
  - High test coverage should be achieved
    - Deep understanding of target code is necessary to write test cases that achieve high coverage

Problems in the Current Industrial Practice (1/2)

- Industry uses many **open source software(OSS)** in their smartphone platforms
  - Samsung's cases: Android(30+ OSS packages), Tizen(40+ OSS packages)

- Most of OSS are shipped in smartphones **without high quality assurance**

Problems in the Current Industrial Practice (2/2)

- Industry does not have enough resources to test open source program code due to time constraints
  - Field engineers **do not have deep knowledge of target program code**
  - Writing effective test cases is a **time-consuming** task

Automated software testing techniques **with modest testing setup effort** to test open source program

Project Scope

- Goal: To **evaluate effectiveness and efficiency** of concolic testing for testing open source programs
- Our team: 1 professor, 2 graduate students, and 1 Samsung Electronics senior engineer
  - Total M/M: 4 persons £ 1 week
- We tested **an open source program `libexif`** used by Samsung smart phones
  - `libexif` consists of 238 functions in C (14KLOC, 3696 branches)
- We used **CREST-BV and KLEE** as concolic testing tools and Coverity Prevent as a static analysis tool
  - We compared CREST-BV and Coverity Prevent in terms of bug detection capability
  - We compared the two concolic testing tools in terms of TC generation speed and bug detection capability

CREST-BV and KLEE

- CREST-BV and KLEE are concolic testing tools
  - They can analyze target C programs
  - They are open source tools

- CREST-BV
  - An extended version of CREST with bit-vector support
  - Instrumentation-based concolic testing tool
    - Insert probes to extract symbolic path formula

- KLEE
  - Implemented on top of the LLVM virtual machine
    - Modify VM to extract symbolic path formula
  - Implements POSIX file system environment model

Effectiveness of Concolic Testing

- Concolic testing is **effective to detect hidden bugs** in open-source programs **with modest effort**
  - We took only **1 week** to **detect 6 crash bugs** in `libexif` without background of the target program
  - Previous case studies
    - Industrial Application of Concolic Testing on Embedded Software: Case Study, ICST 2012
    - Concolic Testing of the Multi-sector Read Operation for Flash Storage Platform Software, FACJ 2012

- Concolic testing was **more effective than static analysis** in this project
  - All the detected bugs were not detected by Coverity Prevent

EXchangeable Image file Format(EXIF)

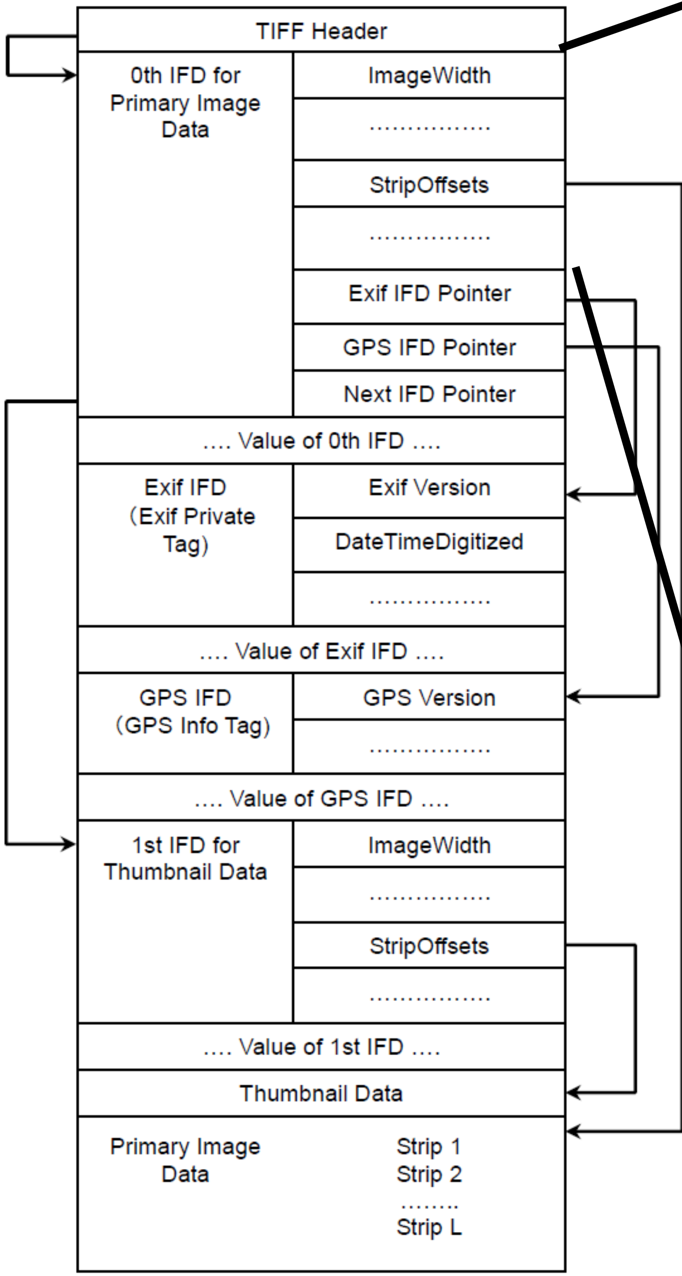
- EXIF is a standard that specifies metadata for image and sound files

Header		
EXIF	Tag	Value
	Width	200
	Height	430
	Date	110522
	…	…
Maker note	Tag	Value
	ISO	200
	Focus	AI Focus
	…	…

- EXIF defines image structure, characteristics, and picture-taking conditions

- Maker note is manufacturer-specific metadata
  - Camera manufactures define a large number of their own maker note tags
  - Ex. Canon has 400+ tags, Fuji has 200+ tags, and so on
  - No standard

xif s



Tag Name	Field Name
A. Tags relating to image data structure	
Image width	ImageWidth
Image height	ImageLength
Number of bits per component	BitsPerSample
Compression scheme	Compression
Pixel composition	PhotometricInterpretation
Orientation of image	Orientation
Number of components	SamplesPerPixel
Image data arrangement	PlanarConfiguration
Subsampling ratio of Y to C	YCbCrSubSampling
Y and C positioning	YCbCrPositioning
Image resolution in width direction	XResolution
Image resolution in height direction	YResolution
Unit of X and Y resolution	ResolutionUnit
B. Tags relating to recording offset	
Image data location	StripOffsets
Number of rows per strip	RowsPerStrip
Bytes per compressed strip	StripByteCounts
Offset to JPEG SOI	JPEGInterchangeFormat
Bytes of JPEG data	JPEGInterchangeFormatLength
C. Tags relating to image data characteristics	
Transfer function	TransferFunction
White point chromaticity	WhitePoint
Chromaticities of primaries	PrimaryChromaticities
Color space transformation matrix coefficients	YCbCrCoefficients
Pair of black and white reference values	ReferenceBlackWhite
D. Other tags	
File change date and time	DateTime
Image title	ImageDescription
Image input equipment manufacturer	Make
Image input equipment model	Model
Software used	Software
Person who created the image	Artist
Copyright holder	Copyright

Test Experiment Setting

- Max time is set to 15, 30 and 60 minutes
- We used `test-mnote.c` in `libexif` as a test driver program
- HW setting
  - Intel Core2duo 3.6 GHz, 16GB RAM running Fedora 9 64bit

Testing Strategies

- Open source oriented approach for test oracles
  - Focusing on runtime failure/crash bugs only
    - Null-pointer dereference, divide-by-zero, out-of-bound memory accesses, etc

- How to setup effective and efficient symbolic input?
  1. Baseline concolic testing
  2. Focus on the maker note tags with concrete image files

Baseline Concolic Testing

- Input EXIF metadata size fixed at 244 bytes
  - Minimal size of a valid EXIF metadata generated by a test program in `libexif`

Header		
EXIF	Tag	Value
	…	…

- 244 bytes long minimal symbolic input file

244 bytes

......

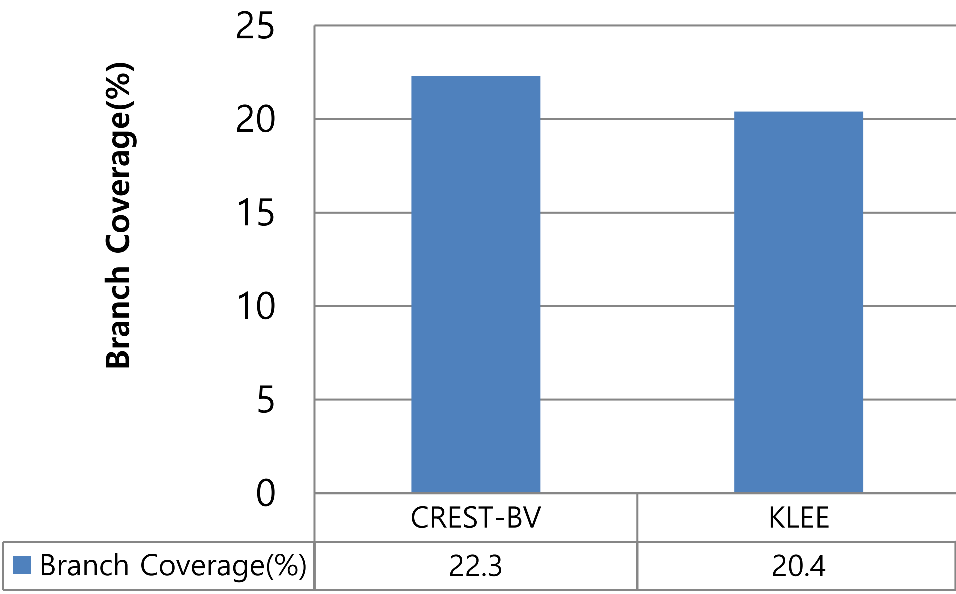
In CREST-BV

```
char array[244];
for (i=0;i<244;i++)
  sym_char(array[i]);
```

Testing Result of Baseline (1/2)

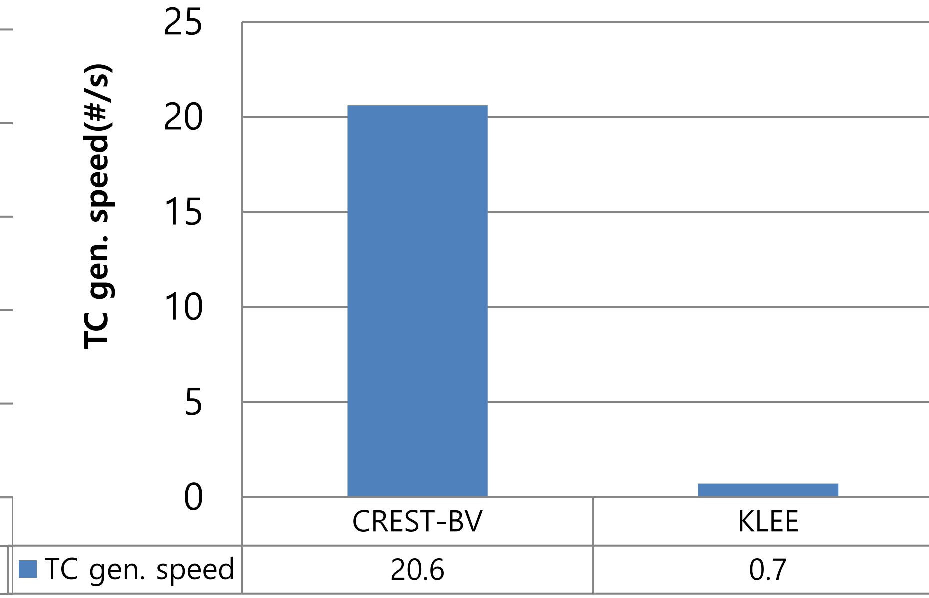
Branch Coverage of CREST-BV and KLEE

(Sum of all search strategies for each tool)



Test case generation speed

(Avg. of the all search strategies for each tool)



- One out-of-bound memory access bug was detected (CVE-2012-2836)

`exif_data_load_data()` in `exif-data.c`

```
1:if (offset + 6 + 2 > ds) { return; }
2:n = exif_get_short(d+6+offset, ...)
```

- KLEE is slower due to
  - Overhead of VM
  - Complex symbolic execution features such as symbolic pointer dereference

Testing Result of Baseline (2/2)

- We analyzed uncovered code to improve branch coverage
  - 5 among 238 functions take 27% of total branches
- Baseline concolic testing could not generate maker notes in a given time
  - We focused on maker notes to improve code coverage

Focus on the Maker Note

- Focus on the maker note tags with concrete image files.
  - We used 6 image files from http://exif.org
  - We used concrete header and standard EXIF metadata and set maker note as symbolic inputs

Header		
EXIF	Tag	Value
	Width	200
	Height	430
	Date	110522
	…	…
Maker note	Tag	Value
	ISO	200
	Focus	AI Focus
	…	…

- Header and standard EXIF metadata are concrete

- Set maker note tags in the image as symbolic inputs

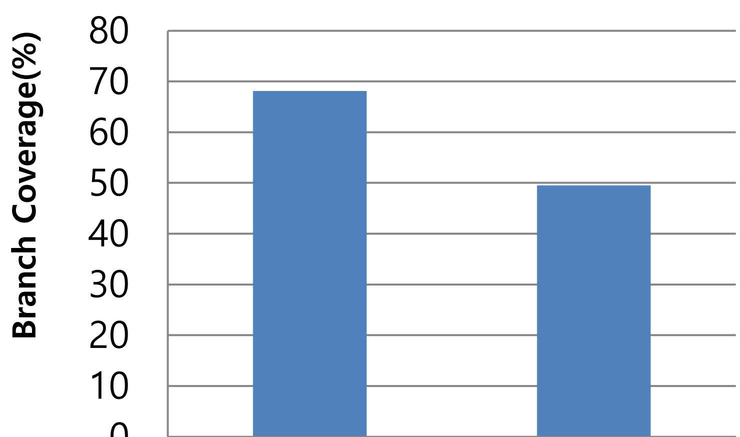
Rationale for the Focus on Maker Note

- We expect that the libexif code that handles maker notes is error-prone due to lack of official specification
- Note that 5 functions among the top 10 largest functions are related to maker notes
  - These 5 functions takes around 27% of total libexif branches

Rank	Function name	# of branches	Cum. # of branches	Cum. # of branches /Total(%)
1	**mnote_olympus_entry_get_value**	508	508	14.3
2	exif_entry_get_value	396	904	25.5
3	exif_entry_initialize	204	1108	31.3
4	**mnote_canon_entry_get_value**	146	1254	35.4
5	**mnote_pentax_entry_get_value**	140	1394	39.4
6	exif_entry_fix	140	1534	43.3
7	**mnote_fuji_entry_get_value**	100	1634	46.1
8	**exif_mnote_data_olympus_load**	96	1730	48.8
9	exif_loader_write	92	1822	51.4
10	exif_data_load_data_content	72	1894	53.5

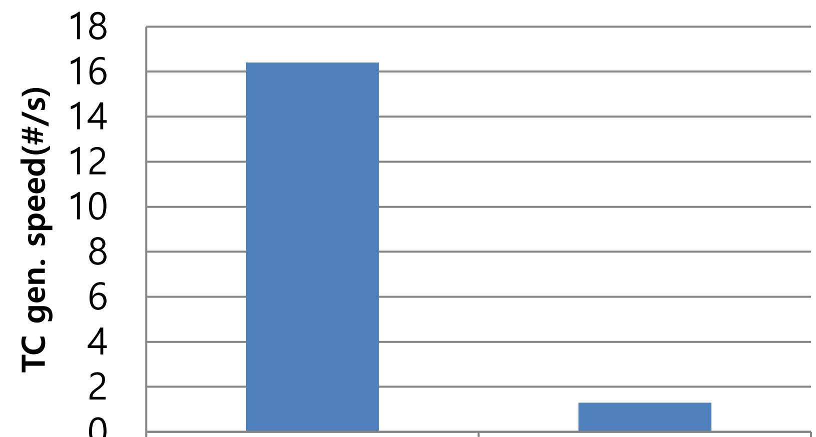
Testing Result of Maker Note (1/2)

Branch Coverage of CREST-BV and KLEE
(Sum of all search strategies for each tool)



	CREST-BV	KLEE
Branch Coverage(%)	68.1	49.5

Test case generation speed
(Avg. of the all search strategies for each tool)



	CREST-BV	KLEE
TC gen. speed	16.4	1.3

- KLEE detected 1 null-pointer-dereference
- CREST-BV detected the null-pointer-dereference bug and 4 divide-by-zero bugs

Testing Result of Maker Note (2/2)

- Null-pointer-dereference bug

mnote_canon_tag_get_description() in mnote-canon-tag.c

```
table[] = { …
      {MNOTE_CANON_TAG_CUSTOM_FUNCS, "CustomFunctions",
        N_("Custom Functions"), ""},
      {0, NULL, NULL, NULL} // Last table entry
…
for(i=0;i<sizeof(table)/sizeof(table[0]);i++)
    //t is a maker note tag read from an image
    if (table[i].tag==t) {
      //Null-pointer dereference occurs when t is 0!!!
      if(!*table[i].description)
        return "";
```

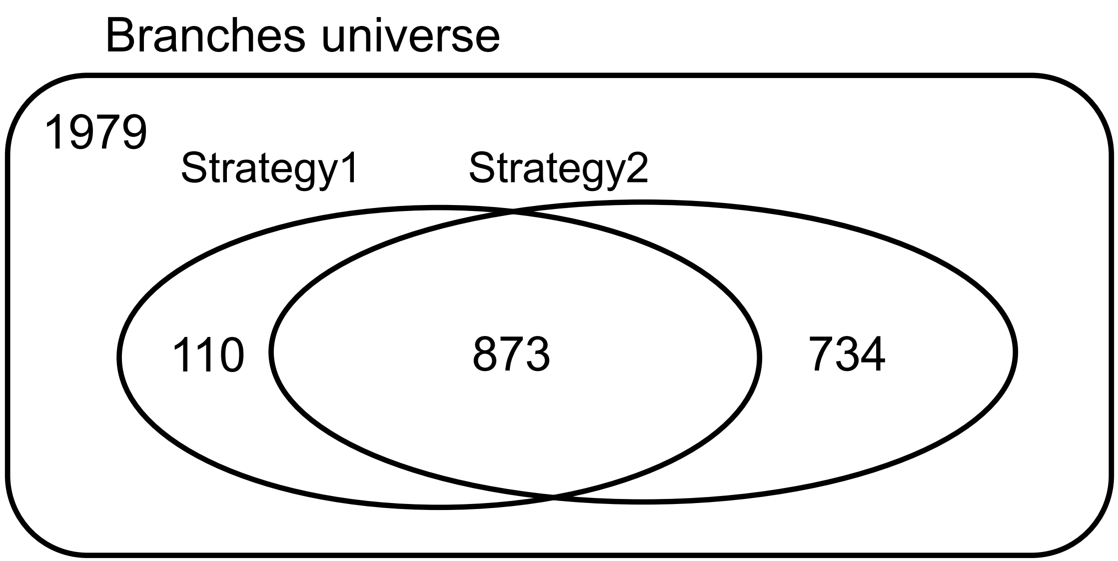
- Divide-by-zero bug (CVE-2012-2837)

mnote_olympus_entry_get_value() in mnote-olympus-entry.c

```
vr=exif_get_rational(...);
//Added for concolic testing
assert(vr.denominator!=0);
a = vr.numerator / vr.denominator;
```

Total result (Baseline + MakerNote)

- Different testing strategies improve coverage
- Total # of covered branches: 1717 (46.5%) among 3696 branches in 1.5 days
  - 110 branches are covered by only the Baseline strategy
  - 734 branches are covered by only the MakerNote strategy
  - 873 branches are covered by both
- In fact, we generated test cases quicker by using multiple machines



Comparison between CREST-BV and Prevent

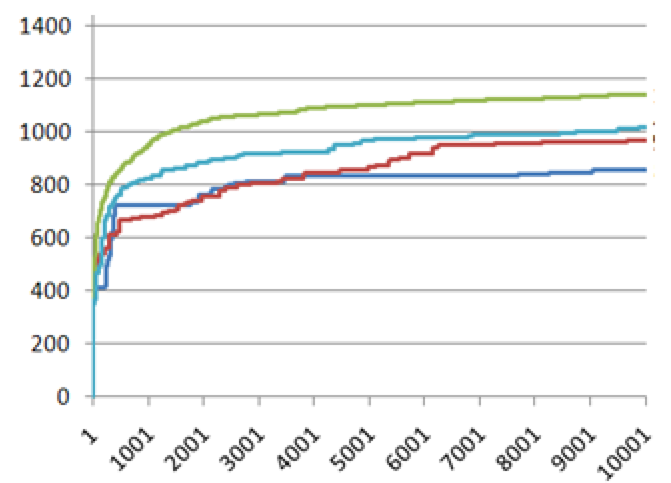
- Prevent failed to detect bugs detected by concolic testing
  - Prevent generated 14 false warnings out of total 15 warnings

- Prevent detected the following null-pointer dereference bug in 5 minutes
  - KLEE/CREST-BV did not detect the bug because our test driver program does not call the buggy function

```
At conditional (1): "!loader" taking the true branch.
CID 10002: Dereference after null check (FORWARD_NULL)
Comparing "loader" to null implies that "loader" might be null.
▲ 413        if (!loader || (loader->data_format == EL_DATA_FORMAT_UNKNOWN)) {
Dereferencing null variable "loader".
▲ 414                exif_log (loader->log, EXIF_LOG_CODE_DEBUG, "ExifLoader",
  415                        "Loader format unknown");
```

Summary of the Challenges

- Libexif is a hard target for concolic testing
  - Hard to specify assertions
    - Requirement specification is very large and complex (182 page official documents + unofficial maker note specifications)
    - Code size is large (14k LOC) and components are hard to understand due to strong connectivity
  - Hard to generate valid inputs
    - Libexif requires strictly structured/formatted input
      - If any one byte of an EXIF header input violates EXIT structure, that entire input is thrown away
  - Search space is very large
    - 10,000 test cases are too little compared to a number of all possible execution paths of a large program such as libexif
    - For example, in another study, 700,000 test cases for `grep` (12k lines) covers only 42% of branches.



Lessons Learned from Real-world Application

- Practical strength of concolic testing
  - 1 null-pointer dereference, 1 out-of-bound memory access, and 4 divide-by-zero in 4 man-weeks
  - Note that
    - `libexif` is very popular OSS used by millions of users
    - we did not have background on `libexif`!!!

- Importance of testing strategy
  - Still state space explosion is a big obstacle
  - Average length of symbolic path formula = 100(baseline strategy)

  => In theory, there can exist 2^{100} different execution paths

- Advantages of CREST-BV over KLEE and Prevent
  - Concolic testing can supplement static analysis