Finding Real Bugs in Big Programs with Incorrectness Logic

Quang Loc Le

University College London

September, 2022

OOPSLA'22

with Azalea Raad, Jules Villard, Josh Berdine, Derek Dreyer, and Peter O'Hearn

First Axiom of a Bug Catching Tool at Scale

"Don't Spam the Developers!"

Interaction with OpenSSL Developers

Pulse-X found 41 bugs, **15 were unknown previously**

- We committed fixes in pull request #15834

```
static int ssl_excert_prepend(SSL_EXCERT **pexc) {
   SSL_EXCERT *exc = app_malloc(sizeof(*exc),
                                "prepend cert");

+  if(exc == NULL)
+    return 0;
   memset(exc, 0, sizeof(*exc));
  ...
}
```

OpenSSL developer:

False positive, app_malloc() *doesn't return if the allocation fails.*

Interaction with OpenSSL Developers - Error trace

```
apps/lib/s_cb.c:959: error: Nullptr Dereference
  PISL found a potential null pointer dereference on line 959.

apps/lib/s_cb.c:957:23: in call to `app_malloc`
  955. static int ssl_excert_prepend(SSL_EXCERT **pexc)
  956. {
  957.     SSL_EXCERT *exc = app_malloc(sizeof(*exc), "prepend cert");
                             ^
  958.
  959.     memset(exc, 0, sizeof(*exc));

test/testutil/apps_mem.c:16:16: in call to `CRYPTO_malloc` (modelled)
    14. void *app_malloc(size_t sz, const char *what)
    15. {
    16.     void *vp = OPENSSL_malloc(sz);
                   ^

test/testutil/apps_mem.c:16:16: is the null pointer
    14. void *app_malloc(size_t sz, const char *what)
    15. {
    16.     void *vp = OPENSSL_malloc(sz);
                   ^
    17.
    18.     return vp;
...
```

Interaction with OpenSSL Developers - *grep* search

another `app_malloc` in `apps/lib/apps.c`

```
void app_bail_out(char *fmt, ...) {
    va_list args;
    va_start(args, fmt);
    BIO_vprintf(bio_err, fmt, args);
    va_end(args);
    ERR_print_errors(bio_err);
    exit(EXIT_FAILURE);
}

void *app_malloc(size_t sz, const char *what) {
    void *vp = OPENSSL_malloc(sz);

    if (vp == NULL)
        app_bail_out("%s: Could not allocate %zu bytes for %s\n",
                     opt_getprog(), sz, what);
    return vp;
}
```

Interaction with OpenSSL Developers - accept fix

apps/lib/s_cb.c Outdated Hide resolved

```
...   ...   @@ -956,6 +956,9 @@ static int ssl_excert_prepend(SSL_EXCERT **pexc)
956   956    {
957   957        SSL_EXCERT *exc = app_malloc(sizeof(*exc), "prepend cert");
958   958
      959   +      if (!exc) {
```

paulidale 13 days ago Contributor

False positive, `app_malloc()` doesn't return if the allocation fails.

lequangloc 13 days ago Author

Our tool recognizes app_malloc() in test/testutil/apps_mem.c rather than the one in apps/lib/apps.c. While the former doesn't return if the allocation fails, the latter does. How do we know which one is actually called?

paulidale 13 days ago Contributor

It would need to look at the link lines or build dependencies to figure out which sources were used.

We should fix the one in `test/testutil/apps_mem.c`.

Then, he created pull request #15836 to commit the fix.

Pulse-X: A bug catching tool

Prove the presence of bugs

- Precision
  - *Doesn't Spam the Developers.*
- Scalability
  - 3-dimensional scale: code (large codebases), people (big team), velocity (high frequency of code changes)
  - continuous integration (CI) reasoning

Pulse-X $\Leftarrow$ Infer

Compositional Shape Analysis by Means of Bi-Abduction (POPL'09)

- analysed Linux Kernel 2.6.25.4 (2.473 MLOC) $<$ 30 mins
- led to Facebook's Infer in 2013[1]

Facebook Acquires Monoidics

MERGERS AND ACQUISITIONS START UP UK

Published on July 18, 2013

Facebook acquired **Monoidics**, a London, UK-based startup that provides a tool for visualizing software quality.

The amount of the deal was not disclosed. Following the close of transaction, the team of the company will join Facebook's office in London.

Founded in 2009 by Italians Dino Distefano (CSO) and Cristiano Calcagno (CTO), and Peter O'Hearn (Scientific Advisor), and led by Bee Lavender (CEO), Monoidics provides INFER, an advanced static code analyzer, which helps users verify their software is bug-free and allows them to focus directly on memory safety and security.

Customers included Airbus, Mitsubishi, ARM, Vanguardistas, and Lawrence Livermore National Laboratory.

[1] http://www.finsmes.com/2013/07/facebook-acquires-monoidics.html

Pulse-X $\Leftarrow$ Infer

Compositional Shape Analysis by Means of Bi-Abduction (POPL'09)

Two concerns:

- Clash with foundations
- Report bugs compositionally

Clash with foundations

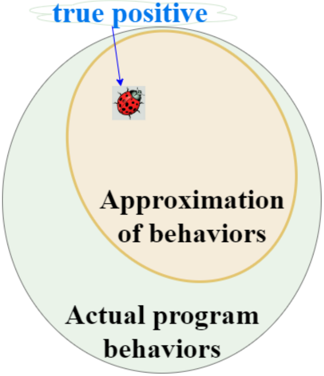
Prove the presence of bugs

Under-approximation vs. Over-approximation

Clash with foundations

Under-approximate reasoning

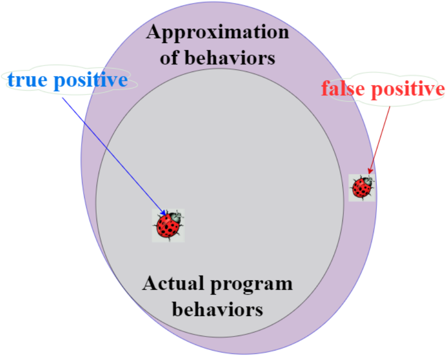
- symbolic execution (KLEE), symbolic model checking (CBMC)
- whole-program analysis
- advantages:
    - report true bugs
- disadvantages:
    - not scaled (for CI)
    - memory model: does not support symbolic heaps



Clash with foundations

Over-approximate reasoning

- compositional reasoning by means of bi-abduction (Infer)
- begin-anywhere analysis
- advantages:
  - scalability
  - memory model: separation logic
- disadvantages:
  - may report false positives



Clash with foundations

Prove the presence of bugs

under-approximate reasoning	over-approximate reasoning
symbolic execution (KLEE), symbolic model checking (CBMC)	compositional reasoning by means of bi-abduction (Infer)
whole-program analysis	begin-anywhere analysis
not scaled	scalability
memory model: does not support symbolic heaps	memory model: separation logic
true bugs	false positives

How to achieve both scalability and precision?

Clash with foundations

A scalable and precise bug-finding tool

- true bugs and scalability
  1. under-approximate analogue of Infer; or
  2. compositional analogue of KLEE, CBMC
- memory model:
  - under-approximate analogue of separation logic
    - $\Rightarrow$ incorrectness separation logic (CAV'20)

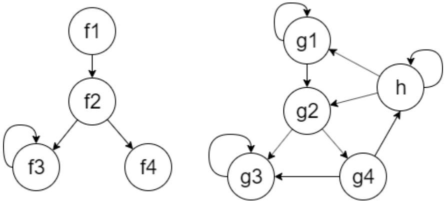
Clash with foundations: This work

an under-approximate analogue of Infer using incorrectness separation logic

Compositional reasoning

The analysis result of a composite program is defined in terms of the analysis results of its parts and a means of combining them.

- part: procedures



- analysis result: under-approximate specs i.e., incorrectness triples[2]
- a means: under-approximate bi-abduction

[2] Peter O'Hearn. Incorrectness Logic. POPL'20

Incorrectness triple[3]

Under-approximate triple

$[P]\ c\ [Q]$ *iff* $post(c)P \supseteq Q$

For all states s in Q, s can be reached by running ***c*** on some s' in P

Incorrectness triple

$[P]\ c\ [\epsilon : Q]$

ϵ: exit condition

- [*ok*: normal execution]
- [*er*: erroneous execution]

[3] Peter O'Hearn. Incorrectness Logic. POPL'20

Incorrectness triple: Examples

Example 1:

Procedure spec: $[y \mapsto Z]$ `free(y)` $[ok\!: y \not\mapsto]$

if y points to a heap cell at the beginning then the cell will be invalidated after executing the `free` procedure.

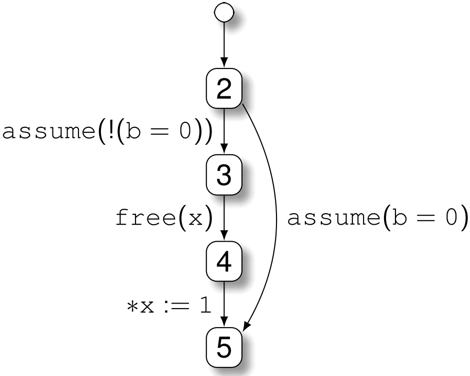
Example 2:

Procedure spec: $[y \not\mapsto]$ `free(y)` $[er\!: y \not\mapsto]$

the spec encodes a double-free error.

Analysis problem

```
void f(bool b, int *x){
  if(b){
   free(x);
   *x := 1;
  }
}
```



Given:

- a program: control flow graphs
- specs of atomic procedures and libraries are given

Question:

- find spec of the program

Under-approximate bi-abduction

Over-approximate bi-abduction question:

$$A * ?M \vdash G * ?F$$

Under-approximate bi-abduction question:

$$A * ?F \vdash G * ?M$$

- abductive inference: find F
- anti-abductive inference: find M

Compositional Bug Reporting: Existing approaches

Without considering the entire program, how do we know a bug is true?

Do you report a null pointer dereference?

```
void f(int* x) {
  *x = 42;
}
```

- Infer uses heuristics:
  - surfacing failed proofs and bug patterns.
- UC-KLEE uses heuristics with annotations
  - OpenSSL-1.0.2: 11 real bugs / 474 errors found $=$ 2.32%
- Pulse-X: $[x \mapsto X * X \mapsto \_]\ \texttt{f(x)}\ [\mathit{ok}\colon x \mapsto X * X \mapsto 42]$

  $[x \mapsto \texttt{null}]\ \texttt{f(x)}\ [\mathit{er}\colon x \mapsto \texttt{null}]$

  $[x \not\mapsto]\ \texttt{f(x)}\ [\mathit{er}\colon x \not\mapsto]$

Compositional Bug Reporting: Pulse-X

```
static int ssl_excert_prepend(SSL_EXCERT **pexc) {
   SSL_EXCERT *exc = app_malloc(sizeof(*exc),
                                "prepend cert");

   memset(exc, 0, sizeof(*exc));
  ...
}
```

Listing 1: OpenSSL null pointer bug in `ssl_excert_prepend`.

Manifest error

- for any value of input `exc`, this error happens.
- any call to `ssl_excert_prepend` will trigger the error.

Compositional Bug Reporting: Pulse-X

```
int chopup_args(ARGS *arg, ...) {
   int num,i;
   ...
   if (arg->count == 0) {
     arg->count=20;
     arg->data= (char **)OPENSSL_malloc(...);
   }
   for (i=0; i<arg->count; i++)
     arg->data[i]=NULL;
   ....
}
```

Listing 2: Latent error in `chopup_args`.

Latent error

- only program paths with inputs `arg->count = 0` lead to error.
- some call to `chopup_args` will trigger the error.

Compositional Bug Reporting: Pulse-X

```
int main(int Argc, char *ARGV[]){
  ARGS arg;
  ...
  arg.count=0;
  ...
  if (!chopup_args(&arg,..)) break;
  ...
}
```

Listing 3: Manifest error in `main` of openssl.c.

Latent error

- only paths with inputs `arg->count = 0` lead to error.
- some call to `chopup_args` will trigger the error.
  - the call in `main`

Compositional Bug Reporting: True Positives Theorem

Theorem (Manifest errors)

An error triple $\models [p]\ \mathsf{C}\ [er : q]$ *with* $q \triangleq \exists \overrightarrow{X_q}.\ \kappa_q \wedge \pi_q$ *denotes a manifest error if:*

1. $p \equiv \mathtt{emp} \wedge \mathtt{true}$;
2. $\mathsf{sat}(q)$ *holds;*
3. $\mathsf{locs}(\kappa_q) \subseteq \overrightarrow{X_q}$, *where* $\mathsf{locs}(.)$ *is the set of heap locations; and*
4. *for all* $\overrightarrow{v}$, $\mathsf{sat}(\pi_q[\overrightarrow{v} / \overrightarrow{Y} \cup \mathsf{locs}(\kappa_q)])$ *holds, where* $\overrightarrow{Y} = \mathsf{flv}(q)$.

$$\mathsf{locs}(\mathtt{emp}) \triangleq \emptyset \quad \mathsf{locs}(x \mapsto X) \triangleq \{x\} \quad \mathsf{locs}(X \mapsto V) = \mathsf{locs}(X \not\mapsto) \triangleq \{X\}$$

$$\mathsf{locs}(\kappa_1 * \kappa_2) \triangleq \mathsf{locs}(\kappa_1) \cup \mathsf{locs}(\kappa_2)$$

Implementation: Scientist vs. Engineer

"Scientists seek perfection and are idealists. ... An engineer's taks is to not be idealistic. You need to be realistic as you have to compromise between conflicting interests." Tony Hoare.

Implementation: with an Incomplete Solver

speed	vs.	precision
dumb but fast	vs.	smart but slow

1. SAT solver: equalities
2. pointer functions, unknown functions

Pulse-X might produce false positives

Evaluation

data set: OpenSSL and 8 open-sourced C++ projects developed and maintained by Facebook.

practical bug classification: for each issue found

- true bug: it has been fixed
- pending bug: the fix has not accepted yet
- false positive: we could not find a fix

$$\text{fix rate} = \text{number of true bugs}/\text{total issues found}$$

Experimental plan:

- run Pulse-X and Infer on each project, collect timings and bugs found
- Scalability: compare the timings
- Precision: check/classify the bugs found on OpenSSL

Evaluation: Goals

- **Hypothesis H1.** On OpenSSL-1.0.1h Pulse-X has a superior fix rate to the present-day Infer.
- **Hypothesis H2.** Pulse-X finds new bugs worth fixing in current OpenSSL.
- **Hypothesis H3.** Pulse-X is broadly comparable with Infer in terms of performance, while reporting a comparable number of bugs.

Evaluation: Summary

New bugs with OpenSSL-3.0.0

- On average, fix rate: Pulse-X: 61% and Infer: 23% - 59%
- Pulse-X found 15 new bugs in OpenSSL-3.0.0
- Pulse-X's performance is as good as Infer's.

Pulse at Facebook: fix rate is 82%.

Take away

Pulse-X: A scalable compositional bug-finding tool

- under-approximate bi-abduction
- true-positives theorem

Experiments, Pulse-X

- found 41 bugs in OpenSSL, 15 were previously unknown.
- fix rate might be 2.7*x* higher than Infer
- as scalable as Infer

Other directions

1. compositional symbolic execution/bounded model checking
2. bug finding tools for concurrent programs
3. backward variant inference for loops
4. test case generation (e.g., with directed fuzz testing)

Evaluation: H1

Old bugs with OpenSSL-1.0.1h

- 8,658 procedures, 444K lines of code, 2.83M of bytes of code
- original Infer found 15 bugs in 2015[4]

Results:

- Pulse-X: 26 issues - 19 true bugs, 7 false positives
  - fix rate: 73%
- Infer: 80 issues - 39 true bugs (8 overlap), 41 false positives
  - fix rate: 48.75%

[4] https://mailing.openssl.dev.narkive.com/2DbkkYzD/openssl-org-3403-null-dereference-and-memory-leak-reports-for-ope

Evaluation: H2

New bugs with OpenSSL-3.0.0

- 22,979 procedures, 754K lines of code, 8.55M of bytes of code

Results:

- Pulse-X: 30 issues - 15 true bugs, 5 pending, 10 false positives
    - fix rate: 50%
    - pull requests: #15834[5], #15836[6], #15910[7],
        - run Pulse-X on the fix, the bug does not occur.
- Infer: 116 issues - 7 true bugs (overlap), 40 false positives, 69 unchecked
    - fix rate: 0.06% - 65%

On average, fix rate: Pulse-X: 61% and Infer: 23% - 59%

[5] https://github.com/openssl/openssl/pull/15834
[6] https://github.com/openssl/openssl/pull/15836
[7] https://github.com/openssl/openssl/pull/15910

Evaluation: H3

Project	#files	LoC(k)	#procs	BoC(m)
OpenSSL-1.0.1h	1536	444	8658	2.83
OpenSSL-3.0.3	2452	754	22979	8.55
wdt	194	25.4	6679	8.5
bistro	424	37.6	7290	9.7
SQuangLe	36	8.3	12938	17.9
RocksDB	1291	411.7	14669	18
FbThrift	5639	937.7	21753	29
OpenR	341	78.3	124461	195.7
Treadmill	409	25.3	236676	393.7
Watchman	557	63.2	245661	407.3

Evaluation: H3