Summaries of C String Loops for More Effective Symbolic Execution (and Refactoring)

Cristian Cadar
Imperial College London

Joint work with
Timotej Kapus (Imperial College London)
Oren Ish-Shalom and Noam Rinetzky (Tel Aviv University)
Shachar Itzhaky (Technion)

Shonan Meeting on Fuzzing and Symbolic Execution
24-27 September 2019
Motivation

● Strings everywhere!
● Lots of work on building string constraint solvers from the SMT community
  ● E.g., Z3, CVC4, HAMPI
● Let’s use them for symbolic execution!
Problem

- Developers often use custom loops instead of string functions

```c
#define whitespace(c) (((c) == ' ') || ((c) == '\t'))
char *p;
for (p = line; p && *p && whitespace(*p); p++)
;
char *p = path + strlen(path);
for (; *p != '/' && p != path; p--)
;
while (*s != '\n')
    s++;
while ('\n' == *beg || '\r' == *beg |
    ('\n' == *beg || '\t' == *beg))
beg++;
Objective

- Replace custom loops with sequence of primitive pointer operations and calls to standard string functions

```c
#define whitespace(c) (((c) == ' ') || ((c) == '\t'))
char *p = line + strspn(line, " \\	")

p = strrchr(path, '/');
p = p == NULL ? path : p;

s = rawmemchr(s, '\n');
pbeg += strspn(pbeg, " \
\n\t");
```
How?

- Counterexample-guided inductive synthesis (based on symex)
- Proof of bounded equivalence (up to a certain string length)
- Mathematical proof of unbounded equivalence
Scope: Memoryless Loops

- Loops conforming to an interface:
  - Argument: single pointer to a string
  - Returns: pointer to an offset in the string
- Only reads the character under current pointer

```c
char* loopSummary(char*);
```
Vocabulary for summarizing string loops

**string.h functions**
- `strspn`
- `strcspn`
- `memchr`
- `strchr`
- `strrchr`
- `strpbrk`

**pointer manipulation**
- `increment`
- `set to start`
- `set to end`

**special**
- `backward traverse`
- `return`

**conditionals**
- `is null`
- `is start`
char *p;
for (p = line; p && *p && whitespace (*p); p++)
    ;

char *p = line + strspn(line, "__\t")
Counter-example guided synthesis

Loop to summarize

Synthesizer

Generate a sequence of tokens fitting all counterexamples

Verifier

Fail - generate counterexample

Success

Done
**Synthesizer**
- Symbolic execution
- Symbolic input: sequence of tokens
- Constrain it to be equivalent on current (counter)examples
- Ask an SMT solver for a solution

**Verifier**
- Symbolic execution
  - Bounded equivalence checking strings of length $\leq 3$
- For memoryless loops:
  - checking lengths $\leq 3$ sufficient to show equivalence for any length (proof in the paper)
  - Intuitively the proof depends on the fact that each iteration is independent from previous ones
Synthesis Evaluation

- 13 open source programs
- Extracted 115 memoryless loops
- 88/115 successfully synthesized within 2h*
- 81 within 5 minutes

*Gaussian process optimization to optimize the vocabulary
Impact of string solvers (KLEE+Z3str) on Sym Ex
Average across loops, 2min timeout

Can reason about unbounded string lengths
Refactoring

- Used summaries to create patches and send them to developers
- Submitted patches to 5 applications
- Patches accepted in libosip, patch and wget

```
- for(; *tmp == ' ' || *tmp == '\t'; tmp++) {
- }
- for(; *tmp == '\n' || *tmp == '\r'; tmp++) {
- }
+ tmp += strspn(tmp, " \t");
+ tmp += strspn(tmp, "\n\r");
```
Conclusion

- C developers often use custom loops to operate on strings.
- Developed synthesis technique to transform such loops into sequences of primitive operations and calls to standard string library.
- Potential to significantly speed up symbolic execution of string-intensive code.
- Applications to refactoring and compiler optimisations.