4 Main Steps of Concolic Testing

1. Instrumentation of a target program
   - To insert probes to build symbolic path formula
2. Transform a constructed symbolic path formula to SMT-compatible format
   - SMT solvers can solve simple formula only
3. Select one branch condition to negate
   - Core technique impacting both effectiveness and efficiency
4. Invoking SMT solvers on the SPF SMT formula
   - Selection of a SMT solver and proper configuration ㅋarameters

4 Main Tasks of Human Engineers

1. Adding proper assert() statements
   - W/o assert(), no test results obtained
2. Selection of symbolic variables in a target program
   - Identify which parts of a target program are most important
3. Construction of symbolic external environment
   - To detect real bugs
4. Performance tuning and debugging
   - To obtain better concolic testing results

Busybox Overview

- We test a busybox by using CREST.
  - BusyBox is a one-in-all command-line utilities providing a fairly complete programming/debugging environment
  - It combines tiny versions of ~300 UNIX utilities into a single small executable program suite.
  - Among those 300 utilities, we focused to test the following 10 utilities
    - **`grep, vi, cut, expr, od , printf, tr, cp, ls, mv.`**
    - We selected these 10 utilities, because their behavior is easy to understand so that it is clear what variables should be declared as symbolic

Experiment overview

- Experimental environments
  - HW: Core(TM)2 E8400@3GHz, 4GB memory
  - OS: fc8 32bit
  - SW: CREST 0.1.1 32bit binary, Yices 1.0.28 32bit library
- Target program: **busybox 1.17.0**
- Strategies: 4 different strategies are used in our experiment.
  - **dfs**: explore path space by (reverse) Depth-First Search
  - **cfg**: explore path space by Control-Flow Directed Search
  - **random**: explore path space by Random Branch Search
  - **random_input**: testing target program by randomly generating input
- In addition, a port-polio approach is applied (i.e., merging the test cases generated by all four above strategies).

Target description -- printf

- Description: print ARGUMENT(s) according to FORMAT, where FORMAT controls the output exactly as in C printf.
- Usage: printf FORMAT [ARGUMENT]...
- Example :
  - input: ./busybox printf ‘%s is coming' 'autumn‘
  - output: autumn is coming

Target program setting -- printf

- Experiment Setting :
  - Target utilities: **busybox printf**
  - Usage: **printf FORMAT [ARGUMENT]...**
  - Symbolic variables setting:
    1. Set **FORMAT** as symbolic value.
       - Type of FORMAT is string. Restrict **5** symbolic characters as input of FORMAT.
    2. Set **ARGUMENT** as symbolic value.
       - Type of ARGUMENT is array of string. Restrict ARGUMENT to **1** length, **10** symbolic characters for each string.
    3. Replace library function by source code: ***strchr().***
  - We perform experiments in the following approach:
    1. run experiment by various strategies.

Result -- printf

Experiment setting:

Iterations: 10,000

branches in printf.c : 144

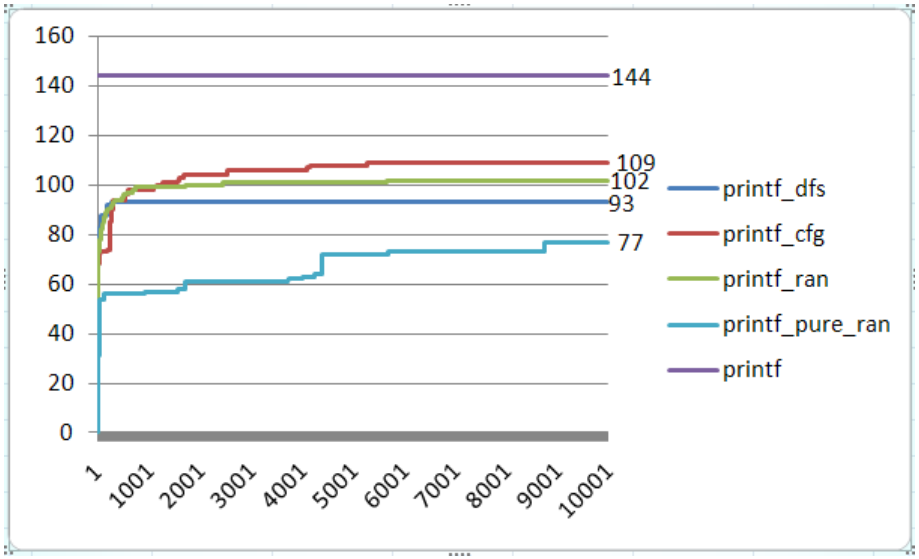
Execution command:

run_crest './busybox '%d123' 0123456789' 10000 -**dfs**

run_crest './busybox '%d123' 0123456789' 10000 -**cfg**

run_crest './busybox '%d123' 0123456789' 10000 -**random**

run_crest './busybox '%d123' 0123456789' 10000 -**random_input**



Strategy	Time cost (s)
Dfs	84
Cfg	41
Random	40
Pure_random	30

Symbolic setup in source code for printf

- Two main instruments in busybox printf.c.
  - Set 2 symbolic inputs: *FORMAT, ARGUMENT.*
  - Replace library function *strchr()* by source code.

```
1.  static void print_direc(char *format, unsigned fmt_length,
2.          int field_width, int precision,
3.          const char *argument)
4.  {
5.  //……
6.  #ifndef CREST
7.      have_width = strchr(format, '*');
8.  #else
9.      have_width = sym_strchr(format, '*');
10. #endif
11. //……
12. }
13. //……
14. int printf_main(int argc UNUSED_PARAM, char **argv)
15. {
16.     int conv_err;
17.     char *format;
18.     char **argv2;
19. //……
20.     format = argv[1];
21.     argv2 = argv + 2;
22.     int i;
23.     int argcc=strlen(format);
24. #ifdef CREST
25.     for( i=0 ; i<argcc ; i++){// argcc = 5 due to the fixed input
26.         CREST_char(format[i]);
27.     }
28.     for(i= 0 ; i<10 ; i++){
29.         CREST_char(argv2[0][i]);
30.     }
31. #endif
32. //……
33. }

34. static char *sym_strchr(const char *str, char ch){

35.     while (*str && *str != ch)
36.         str++;

37.     if (*str == ch)
38.         return str;

39.     return(NULL);

40. }
```

Target description -- grep

- Description: Search for PATTERN in FILEs (or stdin).
- Usage: grep [OPTIONS] PATTERN [FILE]
  - OPTIONS includes

  [-1nqvscFiHhf:Lorm:wA:B:C:EaI] (option followed by ":" means one argument is required.)
- Example :
  - "test_grep.dat" contains

    ```
    define
    enifed
    what is defined?
    def ine
    ```
  - input: `busybox grep "define" test_grep.dat`
  - output:

    ```
    define
    what is defined?
    ```

Options:
-H Add 'filename:' prefix
-h Do not add 'filename:' prefix
-n Add 'line_no:' prefix
-l Show only names of files that match
-L Show only names of files that don't match
-c Show only count of matching lines
-o Show only the matching part of line
-q Quiet. Return 0 if PATTERN is found, 1 otherwise
-v Select non-matching lines
-s Suppress open and read errors
-r Recurse
-i Ignore case
-w Match whole words only
-F PATTERN is a literal (not regexp)
-E PATTERN is an extended regexp
-m N Match up to N times per file
-A N Print N lines of trailing context
-B N Print N lines of leading context
-C N Same as '-A N -B N'
-f FILE Read pattern from file

Instrumentation for Concolic Testing

- Symbolic variable declaration
  - First, identify input variables
  - Second, declare these variables as symbolic variable by inserting CREST_<type>(var_name);
    - If necessary, additional constraints should be given to restrict symbolic variables to have valid ranges of values
      - if (!constraints on var_name) exit(0); shou

Symbolic Variable Declaration for grep

- PATTERN was not declared as symbolic variables, since grep.c handles PATTERN using external binary libraries
  - CREST would not generate new test cases for symbolic PATTERN
  - We used a pattern “define”
- We use a concrete file “test_grep.dat” as a FILE parameter
- Set options as symbolic input (i.e. an array of symbolic character)
  - **23** different options can be given.
    - Specified options are represented by ***option_mask32,*** an ***uint32_t*** value, of which each bit field indicates one option is ON/OFF.
    - Function ***getopt32(char **argv, const char *applet_opts, …)*** is used to generate a bit array ***option_mask32*** which indicates specified options from command line input.
- Set 4 parameters to options as symbolic variables
  - *Copt, max_matches, lines_before, lines_after.*
  - Option argument “***fopt***” is ignored, since it is hard to set file name as symbolic value. “***fopt***”, the parameter of option “-f” (read pattern form an exist file).

Instrumentation in grep.c

```
1. …
2. // before grep_main is called, option_mask32 is
3. // set to represent all command-line options
4. int grep_main (int argc UNUSED_PARAM, char
   **argv)
5. {
6.       getopt32(argv, OPTSTR_GREP,
7.         &pattern_head, &fopt,
8.         &max_matches, &lines_after,
9.         &lines_before,&Copt);
10.      #ifdef CREST
11.      CREST_int(max_matches);
12.      CREST_int(lines_after);
13.      CREST_int(lines_before);
14.      CREST_int(Copt);
15.
16.      #endif
17.      //……
18.      #ifdef CREST
19.      CREST_int(option_mask32);
20.      #endif
21.      if (option_mask32 & OPT_m) {
```

Result of Busybox grep

Experiment 1:

Iterations: 10, 000

branches in grep.c : 178

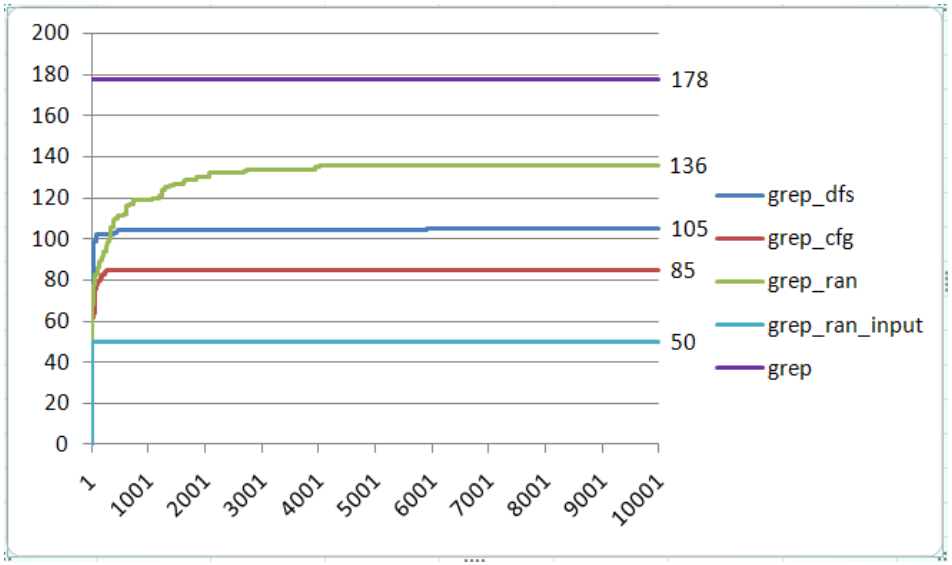
Execution Command:

run_crest './busybox grep "define" test_grep.dat' 10000 -**dfs**

run_crest './busybox grep "define" test_grep.dat' 10000 -**cfg**

run_crest './busybox grep "define" test_grep.dat' 10000 -**random**

run_crest './busybox grep "define" test_grep.dat' 10000 -**random_input**



Strategy	Time cost (s)
Dfs	2758
Cfg	56
Random	85
Pure_random	45

Test Oracles

- In the busybox testing, we do not use any explicit test oracles
  - Test oracle is an orthogonal issue to test case generation
  - However, still violation of runtime conformance (i.e., no segmentation fault, no divide-by-zero, etc) can be checked
- Segmentation fault due to integer overflow detected at grep 2.0
  - This bug was detected by test cases generated using DFS
  - The bug causes segmentation fault when
    - -B 1073741824 (i.e. 2^32/4)
    - PATTERN should match line(s) after the 1st line
    - Text file should contain at least two lines
  - Bug scenario
    - Grep tries to dynamically allocate memory for buffering matched lines (-B option).
    - But due to integer overflow (# of line to buffer * sizeof(pointer)), memory is allocated in much less amount
    - Finally grep finally accesses illegal memory area

Bug 2653 - busybox grep with option -B can cause segmentation fault

Status: RESOLVED FIXED
Reported: 2010-10-02 06:35 UTC by Yunho Kim
Product: Busybox
Modified: 2010-10-03 21:50 UTC (History)
Component: Other
CC List: 1 user (show)
Version: 1.17.x
Platform: PC Linux
Importance: P5 major
Host:
Target Milestone: ---
Target:
Assigned To: unassigned
Build:
URL:
Keywords:
Depends on:
Blocks:
Show dependency tree / graph

Attachments

Add an attachment (proposed patch, testcase, etc.)

Note
You need to log in before you can comment on or make changes to this bug.

Yunho Kim 2010-10-02 06:35:09 UTC

```
I report an integer overflow bug in a busybox grep applet, which causes an
memory corruption.

**** findutils/grep.c ****
634     if (option_mask32 & OPT_C) {
635         /* -C unsets prev -A and -B, but following -A or -B
636            may override it */
637         if (!(option_mask32 & OPT_A)) /* not overridden */
638             lines_after = Copt;
639         if (!(option_mask32 & OPT_B)) /* not overridden */
640             lines_before = Copt;
```

- Bug patch was immediately made in 1 day, since this bug is critical one
  - Importance: P5 major
    - major loss of function
  - Busybox 1.18.x will have fix for this bug

Target description -- vi

- Description: Edit FILE

- Usage: vi [OPTIONS] [FILE] …
- Options:
  - -c Initial command to run ($EXINIT also available)
  - -R Read-only
  - -H Short help regarding available features
- Example :
  - input: cat read_vi.dat

    test for initial command
  - input: cat test_vi.dat

    this is the test for vi

    @#$%&*vi?
  - input: ./busybox vi -c ":read read_vi.dat" test_vi.dat
  - output:

    this is the test for vi
    test for initial command

    @#$%&*vi?

Symbolic Variable Declaration for vi

- We declared a key stroke by a user as a symbolic input character
  - Restrict user key input to **50** symbolic characters.
    - **We modified vi source code so that vi exits after testing 50th key stroke.**
- Set initial command as symbolic input (initial command is only used when option “-c” is specified).
  - Type of initial command is a string (i.e., an array of 17 characters)
- Replace ***4*** library functions with source code: ***strncmp(), strchr(), strcpy(), memchr()***.
- We used a concrete file “test_vi.dat”

4 Functions Added

```
1.  static int sym_strncmp (const char *first,const char *last,int
    count)
2.  {
3.     if (!count)
4.        return(0);
5.
6.     while (--count && *first && *first == *last){
7.          first++;
8.          last++;
9.      }
10.   return( *(unsigned char *)first - *(unsigned char *)last );
11.}
12.
13.static char *sym_strchr(const char *str, char ch){
14.
15.   while (*str && *str != ch)
16.        str++;
17.
18.   if (*str == ch)
19.        return str;
20.
21.   return(NULL);
22.
23.}
24.
```

```
25.static char *sym_strcpy(char *to, const char *from)
26.{
27.        char *save = to;
28.
28.        for (; (*to = *from) != '₩0'; ++from, ++to);
29.        return(save);
30.}
31.
32.void *sym_memchr(const void* src, int c, size_t count)
33.{
34.      assert(src!=NULL);
35.      char *tempsrc=(char*)src;
36.      while(count&&*tempsrc!=(char)c)
37.      {
38.          count--;
39.          tempsrc++;
40.       }
41.       if(count!=0)
42.            return tempsrc;
43.       else
44.            return NULL;
45.}
```

Result of vi

Experiment 1:

Iterations: 10,000

Branches in vi.c : 1498

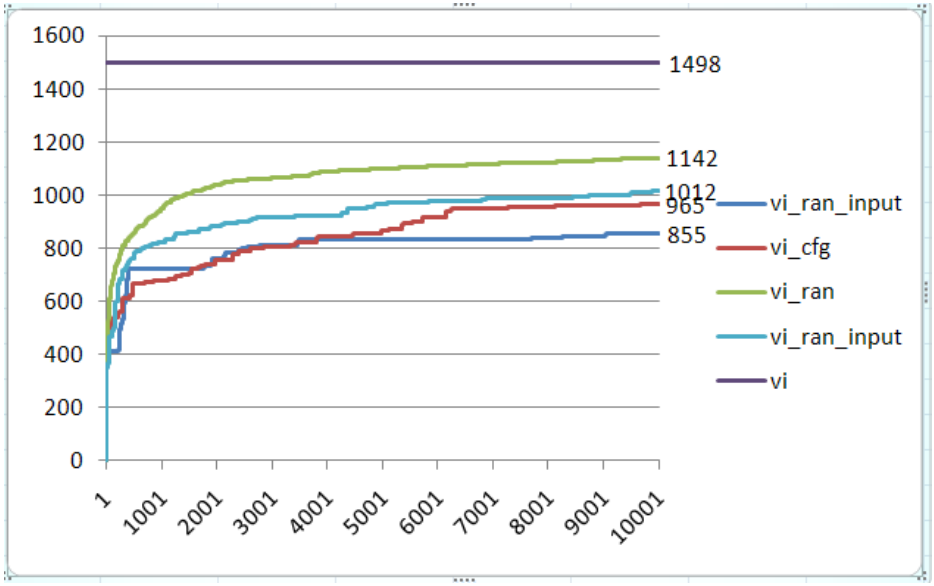
Execution Command:

run_crest './busybox -c ":read read_vi.dat" test_vi.dat' 10000 -**dfs**

run_crest './busybox -c ":read read_vi.dat" test_vi.dat' 10000 -**cfg**

run_crest './busybox -c ":read read_vi.dat" test_vi.dat' 10000 -**random**

run_crest './busybox -c ":read read_vi.dat" test_vi.dat' 10000 -**random_input**



Strategy	Time cost (s)
Dfs	1495
Cfg	1306
Random	723
Pure_random	463