Bug Report from Shadow Guarding

Harish Patil
Computer Sciences Department
University of Wisconsin-Madison
patil@cs.wisc.edu
October 28, 1994

1 Setup

Shadow guarding was implemented on a SPARC 630 MP which is a 4 SPARC processor shared memory machine running SunOs Release 5.3 [UNIX system V release 4]. Sun's multi-threading library[PKB+91] was used to create main and shadow threads. The test programs included integer benchmarks from the SPEC92 test suite, one program from the SafeC test suite and a number of commonly used utilities from SunOs 4.1.3. A utility called *fuzz* [MLB90] was used to generate random input for the SunOs utilities. SPEC benchmarks were tested with their reference inputs.

Utility	Source	# of different errors detected	Program crashes?
cb	SunOs 4.1.3	1	no
col	SunOs 4.1.3	1	yes
decompress	SPEC92	1	no
deroff	SunOs 4.1.3	3	yes
ptx	SunOs 4.1.3	4	no
sc	SPEC92	2	no
ul	SunOs 4.1.3	1	no
uniq	SunOs 4.1.3	1	yes
units	SunOs 4.1.3	1	yes

Run-time errors which do not crash programs can go unnoticed for a long time. Shadow guarding reports such errors promptly after they occur. Programmers can use the feedback from shadow guarding to eliminate subtle bugs. We uncovered unreported errors in five utilities (which did not crash). The SunOs utilities which crashed were already reported to be buggy [MLB90] in earlier versions of SunOs. However, we found new errors in these utilities as well. A complete report on the errors in the utilities tested so far is presented below. We expect further testing will certainly reveal errors in other utilities.

2 Detailed error report

cb

Macros `isop, isalpha, isupper ...` all index the array `_chtype` using character variable '`c`'. The size of the array is 129, with the assumption that input characters will be in the range 1-128. With random input, '`c`' may not be ASCII, violating bounds of array '`_chtype`'.

col

```
229     c3 = *line;
```

Inside the program, there is a lot of places where the pointer '`line`' is incremented without checking its validity.

decompress

Function `getcode()` called from `decompress()`

```
1144   /* high order bits */

1145   code |= (*bp & rmask[bits]) << r_off;
```

Pointer '`bp`' is used to traverse global character array '`buf[BITS]`'.
While decompressing the reference input #of_bits/code changes to 16. (This condition can be forced by changing `#define INIT_BITS` to `16`).
Sometime after this happens, '`bp`' dereferences one location beyond the array '`buf`' at line 1145.

Check: Put '`assert((bp-&buf[0])<BITS);`' before line 1145.

derof

```
341   *++lp = C;

345     *lp = '\0';
```

The character pointer '`lp`' is incremented without being checked. Whenever the line being processed is longer than 512 characters, there will be a pointer out of bound error.

```
812   .. chars[cp[0]]==LETTER
```

Array '`chars`' is indexed using input characters assuming that they are ASCII. For random input many non-ASCII characters result in violation of array bounds for '`chars`'.

ptx

All the errors detected are due to accesses beyond bounds of global arrays. The reason they did not crash the program must be that the errant accesses (illegally) referred to other valid global data.

This program has "`#define isbreak(c) (btable[c])`" where `char btable[128]` is used to indicate whether a character in the range 0-127 is a break character or not. The assumption here is that input characters will all be printable characters in the range 0-127. If the utility is fed a random stream of characters, the check `isbreak(c)` violates the bounds of array btable.

```
332 if(isabreak(*pchar)) {
338 if(isabreak(*pchar++))
```

Function `getline()` uses the pointer `linep` to run through the array "`char line[200]`". `linep` is incremented and dereferenced many times in the function without checking whether input line is small enough to fit in 200 characters.

```
308 *++linep = '\n';
```

Similar error occurs in function `getsort()`

```
469  *linep++ = c;
```

Function `getsort()` has the following code:

```
407       linep = line;
408       while((c = getc(sortptr)) != EOF) {
409           switch(c) {
410
.
415           case '\n':
416               while(isspace(linep[-1]))
```

If `c` is `'\n'` on the very first iteration, `line[-1]` gets accessed.

sc

Found error is `sc.c`, in function `update()`:

```
213       /* Now pick up the counts again */
214       for (i = stcol, cols = 0, col = RESCOL;
215              (col + fwidth[i]) < COLS-1 && i < MAXCOLS; i++) {
```

An array bound violation occurs in the terminating condition of the for loop. The two operands of '`&&`' are in the wrong order '`i < MAXCOLS`' must come before indexing '`fwidth[i]`'.When '`i`' becomes '`MAXCOLS`' (40) there is an array bound violation. (Occurs twice for `input.ref/loada2`).

Found error in file `lex.c`, in function `yylex()`:

```
114      for (tblp = linelim ? experres : statres; tblp->key; tblp++)
115              if (((tblp->key[0]^tokenst[0])&0137)==0
116               && tblp->key[tokenl]==0) {
```

Array '`tokenst`' contains the current token and '`tokenl`' is the length of the current token. The `for` loop traverses a table of reserved words to see if the current token is a reserved word. Pointer '`tblp`' is used to point to various entries of a table of reserved words. The first part of the if condition checks if the first letter of the current reserved word and the current token are the same. (It uses some clever bit manipulation and properties of the ASCII character set to ignore case differences.) The second part makes sure that the current reserved word is not longer than the current token. For input token '`goto`', '`tokenl`' is 4. There is a keyword '`GET`' in the table '`statres`'. The first part of the if condition is satisfied (`(('G'^'g'&0137)==0)` is `TRUE`) For the second check, '`tblp->key[tokenl]`' accesses the 5th element of the current key '`GET`' which has length 4 (`'G','E','T','\0'`). Accessing the 5th element of an array of size 4 is clearly illegal.

ul

There are numerous array bound violations in function filter().
Array 'obuf' is indexed using the variable 'col'. If an input line contains >=512 characters 'col' gets larger than the maximum index (511) allowed for 'obuf'.

units

```
199      *cp++ = c;
```

Pointer "`cp`" is not checked for before it is accessed producing out of bounds pointer reference

uniq

```
76       *buf++ = c;
```

Size of the buffer '`buf`' is not checked before this statement, for input line size greater than 1000, array bound is violated.

References

[MLB90] Barton P. Miller, Fredriksen Lars, and So Brian. An empirical study of the reliability of Unix utilities. *Communications of the ACM*, 33(12):32–44, December 1990.

[PKB+91] M. L. Powell, S. R. Kleiman, S. Barton, D. Shah, D. Stein, and M. Weeks. SunOS multi-thread architecture. In *Proceedings of the Winter USENIX Conference*, pages 1–14, 1991.