For More Information

The full technical report can be found at:

`ftp://grilled.cs.wisc.edu/technical_papers/fuzz-revisited.ps.Z`

The fuzz tools, scripts, test data file, test results, and bug listings can be found at:

`ftp://www.cs.wisc.edu/~paradyn/fuzz-revisited/`

(check the `READ_ME` file)

Discussion

Why aren't companies using this type of testing? It's easy and mostly automatic.

It's good that all of the *re-tested* systems got better. But why not much better?

Why are some of the exact 1990 bugs still present in today's systems?

GUI (X Window) applications are at least as bad as the basic utilties. Should these (more modern) programs be better?

Why are the Linux and GNU results noticeably better than the commerical systems?

- Do they just have an easier job or is there a fundemental difference, such as culture, talent, environment, communications?

Do we require SATAN-like publicity and tools to cause change?

Will the software industry follow the same path as the automobile and steel industries?

Summary of Malloc Test Results

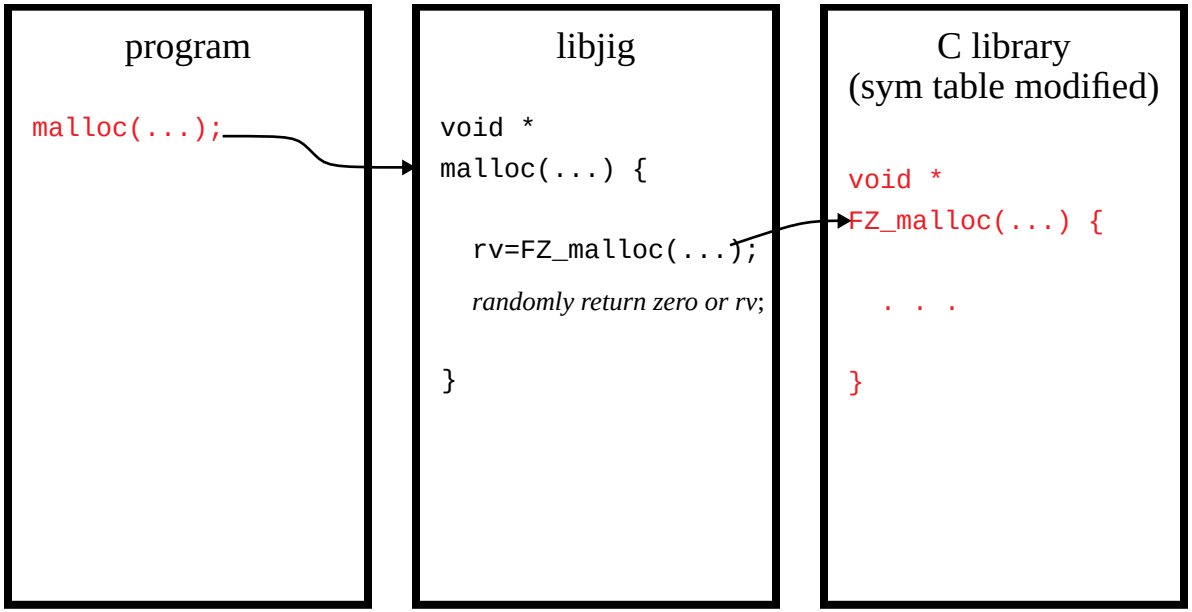
Tested programs in /bin and /usr/ucb on our SunOS 4.1.3 system:

- 53 of these programs used malloc().
- We could crash 25 of the 53 (47%).

Utilities that Crashed				
bar	df	login	rup	tsort
cc	finger	ls	ruptime	users
checknr	graph	man	rusers	vplot
ctags	iostat	mkstr	sdiff	w
deroff	last	rsh	symorder	xsend

Malloc and Friends

Intercept the calls to malloc()



Randomly change the returned value to *zero*: simulating lack of virtual memory.

Malloc and Friends

A common programming error is to not check return values from library and system calls.

Two of the most common areas that we've seen are I/O and memory allocation calls.

As an example, we looked at the use of memory allocation routines:

```
calloc(), malloc(), and realloc()
```

Summary of X Window Test Results

List of Utilities Tested						
bitmap	netscape	xclock	xev	xman	xpostit	xweather
emacs	puzzle	xconsole	xfig	xmh	xsnow	xxgdb
ghostview	rxvt	xcutsel	xfontsel	xminesweep	xspread	
idraw	xboard	xditview	xgas	xneko	xterm	
mosaic	xcalc	xdvi	xgc	xpaint	xtv	
mxrn	xclipboard	xedit	xmag	xpbiff	xv	

X Utility	Input Data Stream Type			
	Random Messages (Type 1)	Garbled Messges (Type 2)	Random Events (Type 3)	Legal Events (Type 4)
# tested	38	38	38	38
# crash/hang	1	10	18	16
%	3%	26%	47%	42%

UNIVERSITY OF WISCONSIN MADISON

Four Types of X Testing

1. *Completely Random Messages:*

   A random series of bytes in a message.

2. *Garbled Messages:*

   Randomly insert, delete, or modify parts of the message stream.

3. *Random Events:*

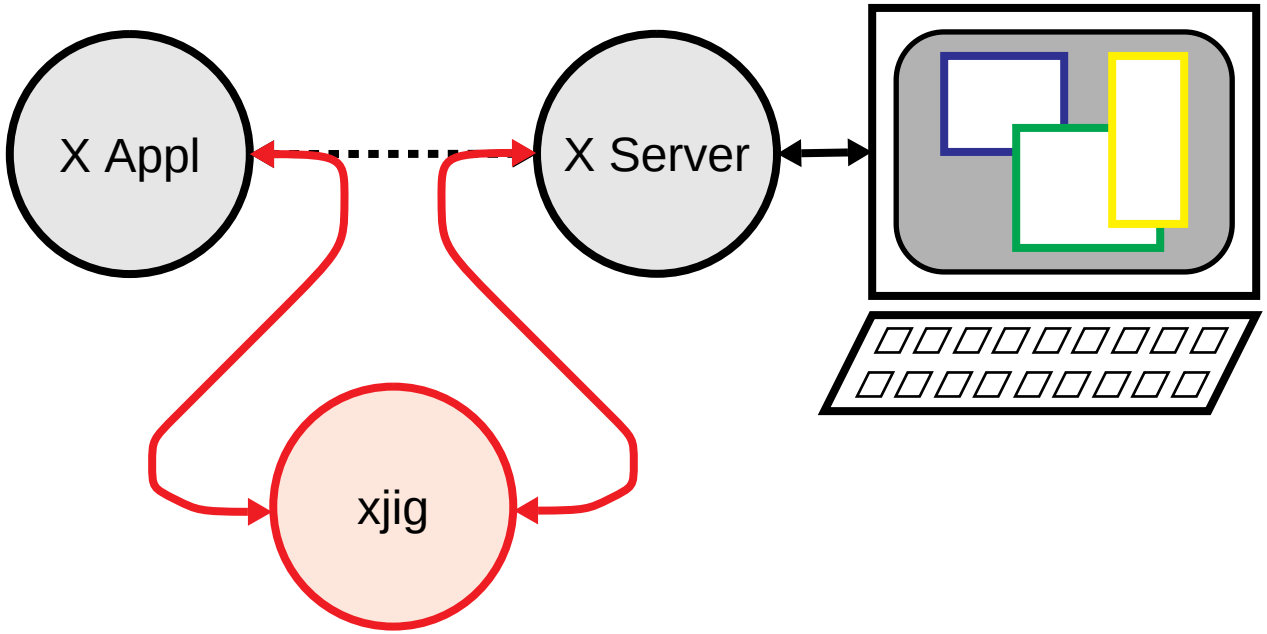
   Keeps track of messageX Protocol message. Randomly insert or modify events with valid size and opcodes. Sequence number, time stamp, and payload may be random.

4. *Legal Events:*

   Protocol conformant messages, logically correct individually and in sequence. Valid values X-Y coordinates, window geometry, parent/child relationships, event time stamps, and sequence numbers.

Intercepting the X Windows Message Stream

We control the messages going to the X application and server by interposing our “xjig” tester.



X Window Applications and Servers

Most modern applications have graphical interfaces, so we didn't want to ignore the X-Window system.

The window display is controlled by a server process.

Communication from the application (client) to the server is by request messages.

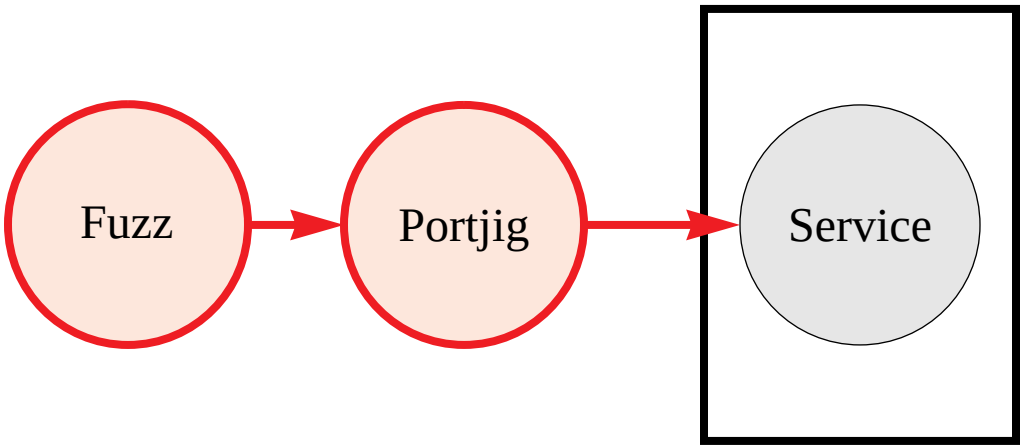
Communication from the server to the application is by events (fixed format messages).

We used the same, simple crash/hang criteria.

Network Services

We tested a wide variety of network services (everything we found in /etc/services).

We connected to and sent random input to each of these services.



We used the same, simple crash/hang criteria.

- Nothing (?) exciting to report: couldn't crash any that we tested!
- We informally did this type of testing 3 to 4 years ago and crashed two services.

Someone seems to be worried enough to do intense testing.

Signed Characters

Number conversions can cause problems;

“char” is a signed, 8-bit integer on (most) UNIX systems.

From eqn (file “lookup.c”):

```
register int h;
...
register char *s = name;

for (h = 0; *s != '\0';)
  h += *s++;
h %= TBLSIZE;
```

The value pointed to by s is a signed character and h is an integer. If a character pointed to
by s may has its high-order bit on, h will be negative.

Dangerous Input Functions

The notorious gets() is still with us!

Multiple occurrences in ftp and telnet, e.g.:

```
gets(line)
```

From the Solaris 2.3 manual page:

> *When using gets(), if the length of an input line exceeds the size of s, indeterminate behavior may result. For this reason, it is strongly recommended that gets() be avoided in favor of fgets().*

Of course, fgets() should be used.

Even new languages, like C++ are not immune:

```
char buff[BUFSIZE];

cin >> buff;
```

You can use “cin.width” function, but this requires extra effort; the default case has dangerous behavior.

Pointer/Array Example 3:

During string processing, from bibtex (file “strpascal.c”):

```
void
null_terminate(s)
char *s;
{
   while (*s != ' ') s++;
   ...
```

Again (!), the loop does not check the size of the array.

Pointer/Array Example 2

Array subscripting error during input, from in cb (file “cb.c”):

```
char string[200];
...
while ((cc = getch()) != c) {
   string[j++] = cc;
   ...
}
```

The termination condition ignores the size of the buffer (`string`).

Pointer/Array Example 1

From ctags in file “ctags.c”:

```
char line[4*BUFSIZ];
...
sp = line;
...
do {
   *++sp = c = getc(inf);
} while ((c != ’\n’) && (c != EOF));
```

Note that the termination condition does not test the size of array (`line`) being used.

Utility	Array/Pointer	Input Functions	Signed Characters	Divide by Zero	EOF Check	Others	No Source Code
join	OU						sN
lex	sShHaUGL						AN
look	shu				HO		N
m4			HU				N
Mail							N
make	h						
nroff				SHOU			AIN
plot	O					sh	N
prolog	sh						
ptx	sShH						A
refer	shU						AHN
spell	sha					SOU	N
spline							N
strings	O						
style						ShH	N
telnet		sShaU					AIN
tsort		sha					N
ul	sShHOUL						AIN
uniq	sShaU						IN
units	SshaO						AIN
vgrind							N

Utility	Array/Pointer	Input Functions	Signed Characters	Divide by Zero	EOF Check	Others	No Source Code
adb	sShHO						
as	a						N
bc							N
bib	S						
cb	haU						AN
cc							N
ccom							ON
checkeq							A
col	O				SU	sha	AIN
colcrt							A
csh						sha	
ctags	O				L		N
dbx	L						s
dc						G	IN
deroff	sShaOU						N
diction						ShHU	N
ditroff	s			SHOU			AN
eqn			sShHU				AIN
ex						h	
fmt							N
ftp		sShaOU					AIN
indent	sh				SHOGL		AN

Programming Errors

The same basic kinds of programming errors found in 1990 dominated the 1995 results:

1. *Array/Pointer*: walking off the end of an array (with either pointers or subscripts) is very popular.

2. *Dangerous input functions*: our friend gets() is still around. And has a C++ cousin!

3. *Signed characters:* the printable ASCII characters need only 7 bits. Dangerous to assume.

4. *Divide (mod) by zero:* the “mod” operator is a divide; must still check the divisor.

5. *EOF checks:* end-of-files do not always occur at convienent places.

6. *Others:* ignoring return codes, importing errors ...

7. No source code: we didn’t always have source code for the crashed utitlities. Sigh.

Summary of Basic Test Results

Utility	SunOS		HP-UX		AIX		Solaris	SGI	Ultrix	NEXT	GNU	Linux
	90	95	90	95	90	95	95	95	95	95	95	95
# tested	77	80	72	74	49	74	70	60	80	75	45	55
# crash/hang	22	18	24	13	12	15	16	9	17	32	3	5
%	29%	23%	33%	18%	24%	20%	23%	15%	21%	43%	7%	9%

UNIVERSITY OF WISCONSIN MADISON

Equivalent Utility Names

Generic Name(s)	SGI	Ultrix	NEXT	GNU	Linux
as			gas	gas	
awk				gawk	
bib/bibtex					
cc			gcc	gcc	
ccom	cfe	cfe	cc1obj	cc1	
compress				gzip	
dbx			gdb	gdb	gdb
ditroff/troff			ptroff		
eqn/deqn		neqn	neqn	geqn	
ex/vi					
lex				flex	flex
plot			psplot		
sh				bash	
soelim				gsoelim	
tbl/dtbl				gtbl	
yacc				bison	

UNIVERSITY OF WISCONSIN MADISON

List of Systems Tested

Identifying Letter	Study	Vendor	Architecture	Kernel
s	1990	Sun Microsystems	Sun 4/110	SunOS 3.2 and 4.0
S	1995		SPARCstation 10/40	SunOS 4.1.3
h	1990	Hewlett Packard	HP 9000/330	4.3 BSD
H	1995		HP 9000/705	HP-UX 9.01
a	1990	IBM	PS/2-80	AIX 1.1
A	1995		RS6000	AIX 3.2
O	1995	Sun Microsystems	SPARCstation 10/40	Solaris 2.3
I	1995	Silicon Graphics	Indy	IRIX 5.1.1.2
U	1995	DEC	DECstation 3100	Ultrix v4.3a rev 146
N	1995	NEXT	Colorstation (MC68040)	NEXTSTEP 3.2
G	1995	GNU, Free Software Foundation		SunOS 4.1.3 & NEXTSTEP 3.2
L	1995	Linux	Cyrix i486	Slackware 2.1.0

So, What Did We Find?

- All three previously-tested versions of UNIX made noticeable improvements, but . . .

  . . . their failure rate is still distressingly high (18-23%).

- Many of the bugs that we found in 1990 are still in the code releases of 1995.
- 15 to 43% of the utilities on the commercial versions of UNIX that (Sun, IBM, SGI, DEC, and NEXT) crashed.
- 9% of the utilities on the freely-distributed Linux version of UNIX crashed (2nd lowest).
- Only 7% of the public GNU utilities crashed (lowest!).
- None of the network services that we tested crashed.
- 3% of X-Window applications crashed on purely random input data streams.

  42% of the applications crash given random, but legal X-event streams.

- None of the X-Window servers that we tested crashed.

Motivation

Our recent experiences hinted that things had not gotten a lot better, so . . .

- We tested lots (9) versions of UNIX, included 3 systems previously tested.
- And what about X-window applications? We tested almost 40 of those. And X servers too.
- And what about network services (like ftpd, rlogind, and telnetd)? We tested those too.
- And we even tested system-library calls to malloc().

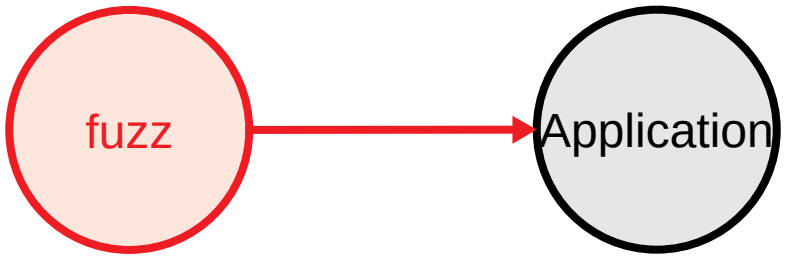
Life had gotten better in some ways, but mostly the results were pretty distressing!

Motivation

In 1990 (see Dec. CACM), we tested almost 90 utility programs on 6 versions of UNIX.

- These systems included: 4.3BSD, SunOS 3.2, SCO's Xenix, IBM 's AOS and AIX 1.1, and Sequent's Dynix 3.0.

Simple (*really*, *really*, simple) automated tests: just feed the programs random inputs:



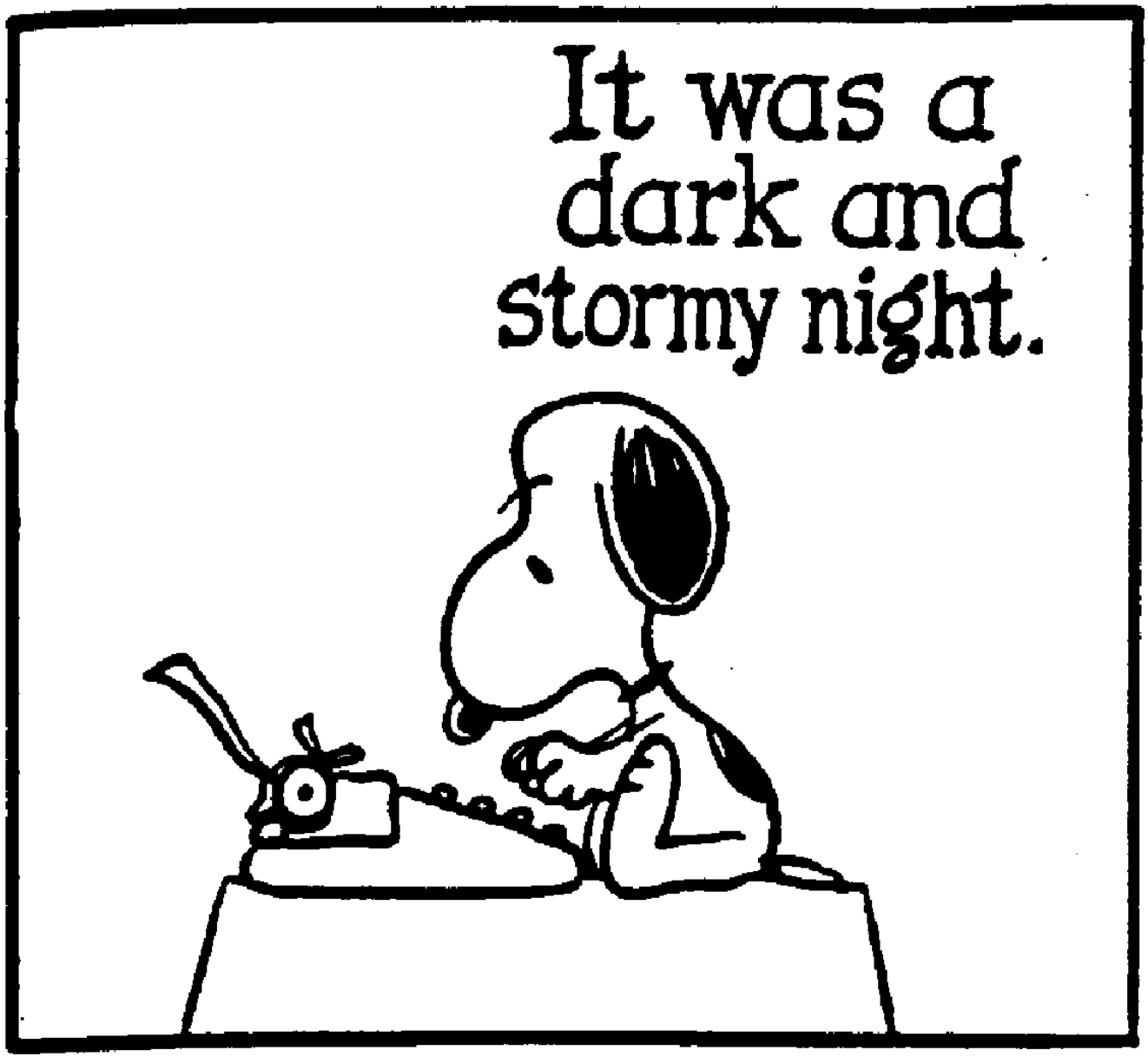
And we could crash or hang from 25% to 33% of the UNIX utilities that we tested.

"Crash" means terminated with core dump.

"Hang" means loops continuous with no apparent end.

VERY SIMPLE reliability criteria!

We also identified and categorized the cause of each crash or hang.



UNIVERSITY OF WISCONSIN MADISON

Making Real Programs Explode: A Simple Application of Random Testing

Barton P. Miller
bart@cs.wisc.edu

Computer Sciences Department
University of Wisconsin
1210 W. Dayton Street
Madison, WI 53706-1685

On leave:

Electrical Engineering Department
Stanford University
Stanford, CA 94305
bart@wis.stanford.edu

UNIVERSITY OF WISCONSIN MADISON