

# The Importance of Benchmarks for Tools that Find or Prevent Buffer Overflows

Richard Lippmann, Michael Zhivich Kendra Kratkiewicz, Tim Leek, Graham Baker, Robert Cunningham

> MIT Lincoln Laboratory lippmann@ll.mit.edu

To be presented at the Workshop on the Evaluation of Software Defect Detection Tools, Co-located with the PLDI 2005 Conference, Chicago 12 June 2005

\*This work was sponsored by the Advanced Research and Development Activity under Force Contract F19628-00-C-0002. Opinions, interpretations, conclusions, and recommendations are those of the authors and are not necessarily endorsed by the United States Government.

**MIT Lincoln Laboratory** 



### Our Experience with Buffer Overflow Detection Tools – Benchmarks are Essential

 An initial literature review led us to believe that tools could reliably find buffer overflows







Automatic Detection of Run-Time Errors at Compile Time A Unique Solution To Reduce Software Testing Costs



Ensuring Flawless Software Reliability

- We created a hierarchy of buffer overflow benchmarks
  - 1. Large full programs
    - Historic versions of BIND, Sendmail, WU-FTP servers with known bufferoverflow vulnerabilities (14)
    - Recent versions of gzip, tar, OpenSSL, Apache
  - 2. 14 Model Programs extracted from servers with known bufferoverflow vulnerabilities (169-1531 lines of code each)

Available from http://www.ll.mit.edu/IST/corpora.html

- 3. 291 Small Diagnostic C Test Cases
  - Created using a buffer overflow taxonomy with 22 attributes, each case varies one attribute

Available from Kendra Kratkiewicz, kendra@ll.mit.edu

**Splint** 



### Model Program Excerpt for Sendmail GECOS Overflow CVE-1999-0131

```
ADDRESS *recipient(...) {
  . . .
  else {
  /* buffer created */
    char nbuf[MAXNAME + 1];
    buildfname(pw->pw gecos,
            pw->pw name, nbuf);
                                          else
                                         /* BAD */
```

```
void buildfname(gecos, login, buf)
  register char *gecos;
  char *login;
  char *buf; {
  register char *bp = buf;
  /* fill in buffer */
  for (p = gecos; *p != ' 0' \&
                   *p != ',' &&
                   *p != ';' &&
                   *p != '%'; p++) {
    if (*p == '&') {
      /* BAD */
      (void) strcpy(bp, login);
      *bp = toupper(*bp);
      while (*bp != ' \setminus 0')
       bp++;
      /* BAD */
      *bp++ = *p;
  *bp = '\0';
```



## **Diagnostic C Test Case Taxonomy**

#### **Taxonomy Attributes**

Attribute Number	Attribute Name	
1	Write/Read	
2	Upper/Lower Bound	
3	Data Type	
4	Memory Location	
5	Scope	-
6	Container	
7	Pointer	
8	Index Complexity	
9	Address Complexity	
10	Length/Limit Complexity	
11	Alias of Buffer Address	
12	Alias of Buffer Index	
13	Local Control Flow	
14	Secondary Control Flow	
15	Loop Structure	
16	Loop Complexity	
17	Asynchrony	
18	Taint	
19	Runtime Environment Dependence	
20	Magnitude	
21	Continuous/Discrete	
22	Signed/Unsigned Mismatch	

#### Scope

Value	Description	
0	same	
1	inter-procedural	
2	global	
3	inter-file/inter- procedural	
4	inter-file/global	

#### Magnitude

*	Value	Description	Example
	0	none	buf[9] = `A';
	1	1 byte	buf[10] = `A';
	2	8 bytes	buf[17] = `A';
	3	4096 bytes	<pre>buf[4105] = `A';</pre>



### OK and BAD (Vulnerable) Diagnostic C Test Case Example

#### **OK Test Case**

#### **BAD (Vulnerable) Test Case**

<pre>'WHEFERAD 0 write 'WHEFERAD 0 urper 'DATA TYPE 0 Char 'BICE BOUND 0 upper 'DATA TYPE 0 Char 'BICE BOUND 0 upper 'DATA TYPE 0 Char 'BICE BOUND 0 upper 'BICE BOUND</pre>	/* Taxonomy Classification: 000100000000000000000	/* Taxonomy Classification: 0001000000000000000000
<pre>"WHCH BOUND 0 upper DATA TYPE 0 char "MEDRY LOCATION 1 heap "BOTTA TYPE 0 char "BOTTA TYPE 0 char "BOTTA TYPE 0 char "DOTTA TYPE 0 char "BOTTA TYPE 0 char "BOTT</pre>	* WRITE/READ 0 write	* WRITE/READ 0 write
<pre>Data type 0 char MEMORY LOCATION 1 heap SCOPE 0 same CONTAINER 0 no POINTER 0 constant TABORY COMPLEXITY 0 constant TODRESS COMPLEXITY 0 constant TODRESS COMPLEXITY 0 voicant TODRESS COMPLEXITY</pre>	* WHICH BOUND 0 upper	* WHICH BOUND 0 upper
<pre>"Mediat DUCATION Theap SCOPE is and "CONTAINER 0 no "NDEX COMPLEXITY 0 constant "ADDRESS COMPLEXITY 0 constant "LENGTH COMPLEXITY 0 NA "ADDRESS COMPLEXITY 0 constant "LENGTH COMPLEXITY 0 NA "LENGTH COMPLEXITY 0 NA "LENGTH COMPLEXITY 0 NA "LENGTH COMPLEXITY 0 NA "LENGTH COMPLEXITY 0 NA "SCOPE STRUCTURE 0 no "LOPE COMPLEXITY 0 NA "SECONDARY CONTROL FLOW 0 none "ECAL CONTROL FLOW 0 none "SECONDARY CONTROL FLOW 0 none "SECONDARY</pre>	* DATA TYPE 0 char	* DATA TYPE O CHAT
<pre>SUPE Usame CONTAINES no POWTER no POWTER no POWTER no POWTER is n</pre>	* MEMORY LOCATION 1 heap	* MEMORI LOCATION I neap
<pre>Converse of the second se</pre>		* SCOFE 0 Same
<pre>PUBLY COMPLEXITY 0 constant ADDRESS COMPLEXITY 0 constant ADDRESS COMPLEXITY 0 constant ADDRESS COMPLEXITY 0 constant ADDRESS ALIAS 0 none LICONTECLENT COMPLEXITY 0 constant ADDRESS ALIAS 0 none LICONTECLE FLOW 0 none SECONDARY CONTROL FLOW 0 none LICOP STRUCTURE 0 no LICOP STRUCTURE 0 no RUNTIME ENV. DEPENDENCE 0 no RUNTIME 0 no verifies CONTINUOUS/DISCRETE 0 discrete SIGNADENESS 0 no Winclude <assert.h> int main(int argc, char *argv[]) { char * buf; buf=(char *)malloc(10*sizeof(char)); assert (buf != NULL); /* OK */ buf[9] = 'A'; return 0;} /* CONTINUO S) </assert.h></pre>		
<pre>* ADDRESS COMPLEXITY 0 constant * ADDRESS COMPLEXITY 0 constant * ADDRESS COMPLEXITY 0 N/A * ADDRESS COMPLEXITY 0 v/A * ADDRESS ALLAS 0 none * INDEX AL</pre>	* POINTER U NO	* INDEX COMPLEXITY 0 constant
<pre>ADDRESS COMPLEXITY 0 NAA ADDRESS ALLAS 0 none ADDRESS ADDRESS ADDRESS ALLAS 0 none ADDRESS ADDRESS ADDRESS ALLAS 0 none ADDRESS ADDRESS ADDRESS ALLAS 0 none ADDRESS ADDRESS ADDRESS ADDRESS ADDRESS ALLAS 0 none ADDRESS ALLAS 0 none ADDRESS ALLAS 0 none ADDRESS ADDRESS ADDRESS AD</pre>	* ADDRESS COMPLEXITY 0 constant	* ADDRESS COMPLEXITY 0 Constant
<pre>ADDRESS ALIAS 0 none * ADDRESS ALIAS 0 none * A</pre>		* LENGTH COMPLEXITY 0 N/A
<pre>* INDEX ALAS 0 none * INDEX ALAS 0 none * LOCAL CONTROL FLOW 0 none * LOOP STRUCTURE 0 no * LOOP COMPLEXITY 0 N/A * ASYNCHRONY 0 no * ASYNCHRONY 0 no * RUNTIME ENV. DEPENDENCE 0 no * CONTINUOUS/DISCRETE 0 discrete * SIGNEDNESS 0 no */ # include <assert.h> int main(int argc, char *argv[]) { { /* OK */ buf=(char *)malloc(10*sizeof(char)); assert (buf != NULL); /* OK */ buf[]0] = 'A'; return 0;} </assert.h></pre>	* ADDRESS ALLAS O nono	* ADDRESS ALIAS 0 none
<pre>LOCAL CONTROL FLOW 0 none SECONDARY CONTROL FLOW 0 none LOOP STRUCTURE 0 no LOOP STRUCTURE 0 no SECONDARY CONTROL FLOW 0 none SECONDARY CONTROL FLOW 0 none LOOP STRUCTURE 0 no SECONDARY CONTROL FLOW 0 none SECONDARY CONTROL FLOW 0 none LOOP STRUCTURE 0 no LOOP STRUCTURE 0 no TAINT 0 no SECONDARY CONTROL FLOW 0 none LOOP STRUCTURE 0 no TAINT 0 no SECONDARY CONTROL FLOW 0 none LOOP STRUCTURE 0 no TAINT 0 no SECONDARY CONTROL FLOW 0 none LOOP STRUCTURE 0 no TAINT 0 no SECONDARY CONTROL FLOW 0 none LOOP STRUCTURE 0 no TAINT 0 no SECONDARY CONTROL FLOW 0 none SECONDARY CONTROL FLOW 0 none SECONTROL FLOW 0 no SECONTROL FLOW 0 NO</pre>	* INDEX ALIAS 0 none	* INDEX ALIAS 0 none
<pre>*SECONDARY CONTROL FLOW 0 none *LOOP STRUCTURE 0 no *LOOP STRUCTURE 0 no *INTIME ENV. DEPENDENCE 0 no *J *J SIGNEDNESS 0 no */ #IINCLUDE <structure 0="" no<br="">*LOOP STRUCTURE 0 no *J SIGNEDNESS 0 no */ #IINCLUDE <structure 0="" no<br="">*LOOP STRUCTURE 0 no *IINT NE ENV. DEPENDENCE 0 no */ #IINCLUDE <structure 0="" no<br="">*/ #IINCLUDE STRUCTURE 0 no */ #IINCLUDE STRUCTURE 0 no */</structure></structure></structure></structure></structure></structure></structure></structure></structure></structure></structure></structure></structure></structure></structure></structure></structure></pre>	* LOCAL CONTROL ELOW () pope	* LOCAL CONTROL FLOW 0 none
<pre>*LOOP STRUCTURE 0 no *LOOP STRUCTURE 0 no *LOOP COMPLEXITY 0 N/A *ASYNCHRONY 0 no *TAINT 0 no *ASYNCHRONY 0 no *TAINT 0 no *MAGNITUDE 0 no overflow *CONTINUOUS/DISCRETE 0 discrete *SIGNEDNESS 0 no */ #include <stdlib.h> #include <stdlib.h> #include <assert.h> int main(int argc, char *argv[]) { char * buf; buf=(char *)malloc(10*sizeof(char)); assert (buf != NULL); /* OK */ buf[9] = 'A'; return 0;}</assert.h></stdlib.h></stdlib.h></pre> *LOOP STRUCTURE 0 no *LOOP STRUCTURE 0 n	* SECONDARY CONTROL ELOW 0 none	* SECONDARY CONTROL FLOW 0 none
<pre>'LOOP COMPLEXITY 0 N/A 'ASYNCHRONY 0 no 'ASYNCHRONY 0 no 'AUNTIME ENV. DEPENDENCE 0 no 'WAGNITUDE 0 no verifiev 'CONTINUOUS/DISCRETE 0 discrete 'SIGNEDNESS 0 no '/ #include <stdlib.h> #include <stdlib.h> #include <stsert.h> int main(int argc, char *argv[]) {     char * buf;     buf=(char *)malloc(10*sizeof(char));     assert (buf != NULL);     /* OK */     buf[9] = 'A';     return 0;} </stsert.h></stdlib.h></stdlib.h></pre>	* LOOP STRUCTURE 0 no	* LOOP STRUCTURE 0 no
<pre>* ASYNCHRONY 0 no * TAINT 0 no * TAINT 0 no * ARYNTIME ENV. DEPENDENCE 0 no * MAGNITUDE 0 no overflow * CONTINUOUS/DISCRETE 0 discrete * SIGNEDNESS 0 no */ #include <stdlib.h> #include <assert.h> int main(int argc, char *argv[]) { char * buf; buf=(char *)malloc(10*sizeof(char)); assert (buf != NULL); /* OK */ buf[9] = 'A'; return 0;} * ASYNCHRONY 0 no * TAINT 0 no * ANYNTIME ENV. DEPENDENCE 0 no * MAGNITUDE 1 byte c CONTINUOUS/DISCRETE 0 discrete * SIGNEDNESS 0 no */ #include <assert.h> int main(int argc, char *argv[]) { char * buf; buf=(char *)malloc(10*sizeof(char)); assert (buf != NULL); /* OK */ buf[9] = 'A'; return 0;}</assert.h></assert.h></stdlib.h></pre>	* LOOP COMPLEXITY 0 N/A	* LOOP COMPLEXITY 0 N/A
<pre>*TAINTO no *TAINTO no *RUNTIME ENV. DEPENDENCE 0 no *RUNTIME ENV. ENV. ENV. ENV. ENV. ENV. ENV. ENV</pre>	* ASYNCHRONY 0 no	* ASYNCHRONY 0 no
<pre>* RUNTIME ENV. DEPENDENCE 0 no * MAGNITUDE 1 no overflow * AGNITUDE 0 no overflow * CONTINUOUS/DISCRETE 0 discrete * SIGNEDNESS 0 no */ #include <stdlib.h> #include <stdlib.h> #include <stdlib.h> #include <assert.h> int main(int argc, char *argv[]) { char * buf; buf=(char *)malloc(10*sizeof(char)); assert (buf != NULL); /* OK */ buf[9] = 'A'; return 0;} </assert.h></stdlib.h></stdlib.h></stdlib.h></pre>	* TAINT 0 no	* TAINT 0 no
<pre>* MAGNITUDE 0 no overflow * CONTINUOUS/DISCRETE 0 discrete * SIGNEDNESS 0 no */ #include <stdlib.h> #include <stdlib.h> #include <assert.h> int main(int argc, char *argv[]) { char * buf; buf=(char *)malloc(10*sizeof(char)); assert (buf != NULL); /* OK */ buf[9] = 'A'; return 0;} MAGNITUDE 1 l byte * MAGNITUDE 1 l byte * CONTINUOUS/DISCRETE 0 discrete * SIGNEDNESS 0 no */ #include <assert.h> int main(int argc, char *argv[]) { char * buf; buf=(char *)malloc(10*sizeof(char)); assert (buf != NULL); /* OK */ buf[9] = 'A'; return 0;} } } *********************</assert.h></assert.h></stdlib.h></stdlib.h></pre>	* RUNTIME ENV. DEPENDENCE 0 no	* RUNTIME ENV. DEPENDENCE 0 no
<pre>'CONTINUOUS/DISCRETE 0 discrete 'SIGNEDNESS 0 no '' #include <stdlib.h> #include <stdlib.h> #include <assert.h> int main(int argc, char *argv[]) {     char * buf;     buf=(char *)malloc(10*sizeof(char));     assert (buf != NULL);     /* OK */     buf[9] = 'A';     return 0;} </assert.h></stdlib.h></stdlib.h></pre>	* MAGNITUDE 0 no overflow	* MAGNITUDE 1 1 byte
<pre>'SIGNEDNESS 0 no '' #include <stdlib.h> #include <assert.h> int main(int argc, char *argv[]) {     char * buf;     buf=(char *)malloc(10*sizeof(char));     assert (buf != NULL);     /* OK */     buf[9] = 'A';     return 0;} </assert.h></stdlib.h></pre>	* CONTINUOUS/DISCRETE 0 discrete	* CONTINUOUS/DISCRETE 0 discrete
<pre>// #include <stdlib.h> #include <stdlib.h> #include <assert.h> int main(int argc, char *argv[]) {     char * buf;     buf=(char *)malloc(10*sizeof(char));     assert (buf != NULL);     /* OK */     buf[9] = 'A';     return 0;}     /* Imclude <stdlib.h> #include <assert.h> int main(int argc, char *argv[])     {         char * buf;         buf=(char *)malloc(10*sizeof(char));         assert (buf != NULL);         /* BAD */         buf[10] = 'A';         return 0;} </assert.h></stdlib.h></assert.h></stdlib.h></stdlib.h></pre>	* SIGNEDNESS 0 no	* SIGNEDNESS 0 no
<pre>#include <stdlib.h> #include <assert.h> int main(int argc, char *argv[]) {     char * buf;     buf=(char *)malloc(10*sizeof(char));     assert (buf != NULL);     /* OK */     buf[9] = 'A';     return 0;}     #include <stdlib.h> #include <assert.h>     int main(int argc, char *argv[])     {         char * buf;         buf=(char *)malloc(10*sizeof(char));         assert (buf != NULL);         /* BAD */         buf[10] = 'A';         return 0;} </assert.h></stdlib.h></assert.h></stdlib.h></pre>	*/	*/
<pre>#include <assert.h> int main(int argc, char *argv[]) {     char * buf;     buf=(char *)malloc(10*sizeof(char));     assert (buf != NULL);     /* OK */     buf[9] = 'A';     return 0;}     #include <assert.h>     int main(int argc, char *argv[])     {         char * buf;         buf=(char *)malloc(10*sizeof(char));         assert (buf != NULL);         /* BAD */         buf[10] = 'A';         return 0;} </assert.h></assert.h></pre>	<pre>#include <stdlib.h></stdlib.h></pre>	<pre>#include <stdlib.h></stdlib.h></pre>
<pre>int main(int argc, char *argv[]) {     char * buf;     buf=(char *)malloc(10*sizeof(char));     assert (buf != NULL);     /* OK */     buf[9] = 'A';     return 0;}     int main(int argc, char *argv[])     {         char * buf;         buf=(char *)malloc(10*sizeof(char));         assert (buf != NULL);         /* BAD */         buf[10] = 'A';         return 0;}</pre>	<pre>#include <assert.h></assert.h></pre>	<pre>#include <assert.h></assert.h></pre>
<pre>int main(int arge, char *argv[]) {   char * buf;   buf=(char *)malloc(10*sizeof(char));   assert (buf != NULL);   /* OK */   buf[9] = 'A';   return 0;}   int main(int arge, char *argv[])   {     {         char * buf;         buf=(char *)malloc(10*sizeof(char));         assert (buf != NULL);         /* BAD */         buf[10] = 'A';         return 0;} </pre>	int main (int and then temper[])	int main(int argc, char *argv[])
<pre>{     char * buf;     buf=(char *)malloc(10*sizeof(char));     assert (buf != NULL);     /* OK */     buf[9] = 'A';     return 0;} </pre> {     {         char * buf;         buf=(char *)malloc(10*sizeof(char));         assert (buf != NULL);         /* BAD */         buf[10] = 'A';         return 0;}	Int main(int arge, char ^argv[])	
<pre>char * buf; buf=(char *)malloc(10*sizeof(char)); assert (buf != NULL); /* OK */ buf[9] = 'A'; return 0;} char * buf; buf=(char *)malloc(10*sizeof(char)); assert (buf != NULL); /* BAD */ buf[10] = 'A'; return 0;}</pre>	{	
<pre>buf; buf; buf; buf; buf=(char *)malloc(10*sizeof(char)); assert (buf != NULL); /* OK */ buf[9] = 'A'; return 0;} buf=(char *)malloc(10*sizeof(char)); assert (buf != NULL); /* BAD */ buf[10] = 'A'; return 0;}</pre>	char * buf:	char * buf;
<pre>buf=(char *)malloc(10*sizeof(char)); assert (buf != NULL); /* OK */ buf[9] = 'A'; return 0;}</pre> buf=(char *)malloc(10*sizeof(char)); assert (buf != NULL); /* BAD */ buf[10] = 'A'; return 0;}		$h_{1}f_{-}(ah_{2}r_{+})m_{2}llog(10*aigoof(ah_{2}r_{+}))$
<pre>assert (buf != NULL); /* OK */ buf[9] = 'A'; return 0;} assert (buf != NULL); /* BAD */ buf[10] = 'A'; return 0;}</pre>	<pre>but=(char *)malloc(10*sizeof(char));</pre>	
<pre>/* OK */ buf[9] = 'A'; return 0;} </pre>	aggert (buf l= NIILL).	assert (buf != NULL);
<pre>/* OK */ buf[9] = 'A'; return 0;} </pre>		
<pre>/* OK */ buf[9] = 'A'; return 0;} </pre>		/* BAD */
<pre>buf[9] = 'A'; return 0;} </pre> <pre>buf[10] = 'A'; return 0;} </pre>	/* OK */	
return 0;}	buf[9] = 'A';	buf[10] = 'A';
return 0;}		
return 0;}		return 0.1
	return 0;}	

### Evaluating Static Analysis Tools with Model Programs and Test Cases



#### **14 MODEL PROGRAMS**

- Most tools can't handle real server code!
- They also exhibit poor performance on extracted model programs
  - Low detection and high false alarm rates
  - Only Polyspace is better than guessing

#### **291 Diagnostic Test Cases**



- Good performance for Archer and Polyspace on simple test cases but
  - Run time for Polyspace is more than two days
  - Archer doesn't perform inter-procedural analysis or handle string functions

## Evaluating Dynamic Test Instrumentation Tools with Benchmarks



# • Some tools accurately detect most overflows in model programs

- CCured, TinyCC, CRED
- Misses are caused by errors in implementation or limited analyses

#### **Increase in Run Time Compared to GCC**



- Some tools can't compile large programs (e.g. CCured, TinyCC, )
- Some tools exhibit excessive (x100) increases in run time (e.g. Chaperon, Insure)
- Only CRED combines good detection with reasonable run times.



## Why Do Remotely Exploitable Buffer Overflows Still Exist?



 As many new buffer overflow vulnerabilities are being found each year today in important internet software as were being found six years ago



## Speech Recognition Benchmarks Led to Dramatic Performance Improvements





- 1969 Mad inventors and untrustworthy engineers, no progress, work has been an experience with no knowledge gained (Pierce, 1969)
- 1981 First publicly available speech data base (Doddington, 1981)
- Today Dramatic progress and many deployed speech recognizers, major focus on corpora and benchmarks (Pallet, 2004)

MÍT Lincoln Laboratory



## Comments

- Don't shoot the messenger
  - It is essential to benchmark tool performance
  - How else can you know how well an approach works and set expectations for tool users?
  - How else can you obtain diagnostic information that can be used to guide further improvements?



- Benchmarks should be fair, comprehensive and appropriate
  - Provide ground truth, measure detection and false alarm rates, run times, memory requirements, ...
  - Include tasks appropriate for the tool being evaluated



- Using tools that "find hundreds of bugs on ..." may be detrimental because they provide a false sense of security
  - What are their detection and miss rates?
  - Are these the type of bugs that we really care about?
- Developers have to think more about how tools fit into the code development/use lifecycle



## References

- Doddington, G. R. and T. B. Schalk (1981). Speech Recognition: Turning Theory into Practice,. <u>IEEE Spectrum</u>,: 26-32.
- Kratkiewicz, K. J. and R. Lippmann (2005). Using a Diagnostic Corpus of C Programs to Evaluate Buffer Overflow Detection by Static Analysis Tools. <u>Workshop on the Evaluation of Software Defect Detection Tools</u>.
- Kratkiewicz, K. J. (2005). Evaluating Static Analysis Tools for Detecting Buffer Overflows in C Code. ALM in IT Thesis in the Harvard University Extension Program.
- Pallett, D. S. (2003). A Look at NIST's Benchmark ASR Tests: Past, Present, and Future, http://www.nist.gov/speech/history/pdf/NIST\_benchmark\_ASRtests\_2003.pdf
- Pierce, J. (1970). "Whither speech recognition?" <u>Journal of the Acoustical</u> <u>Society of America</u> 47(6): 1616-1617.
- Zhivich, M., T. Leek, et al. (2005). Dynamic Buffer Overflow Detection. Workshop on the Evaluation of Software Defect Detection Tools.
- Zitser, M., R. P. Lippmann, et al. (2004). Testing Static Analysis Tools Using Exploitable Buffer Overflows From Open Source Code. <u>Proceedings ACM</u> <u>Sigsoft 2004/FSE Foundations of Software Engineering Conference,</u> <u>http://www.ll.mit.edu/IST/pubs/04\_TestingStatic\_Zitser.pdf.</u>