

Statically Detecting Likely Buffer Overflow Vulnerabilities

David Larochelle

David Evans



Supported by USENIX Student Grant and NASA LRC

University of Virginia
Department of Computer Science

- 1988: Morris worm exploits buffer overflows in fingerd to infect 6,000 servers
- 2001: Code Red exploits buffer overflows in IIS to infect 250,000 servers
 - Single largest cause of vulnerabilities in CERT advisories
 - Buffer overflow threatens Internet- WSJ(1/30/01)



16 August 2001
David Larochelle

2

Why aren't we better off than we were 13 years ago?

- Ignorance
- C is difficult to use securely
 - Unsafe functions
 - Confusing APIs
- Even security aware programmers make mistakes.
- Security Knowledge has not been codified into the development process

16 August 2001
David Larochelle

3

Automated Tools

- Run-time solutions
 - StackGuard[USENIX 7], gcc bounds-checking, libsafe[USENIX 2000]
 - Performance penalty
 - Turns buffer overflow into a DoS attack
- Compile-time solutions - static analysis
 - No run-time performance penalty
 - Checks properties of all possible executions

16 August 2001
David Larochelle

4

Design Goals

- Tool that can be used by typical programmers as part of the development process
 - Fast, Easy to Use
- Tool that can be used to check legacy code
 - Handles typical C programs
- Encourage a proactive security methodology
 - Document key assumptions

16 August 2001
David Larochelle

5

Our approach

- Document assumptions about buffer sizes
 - Semantic comments
 - Provide annotated standard library
 - Allow user's to annotate their code
- Find inconsistencies between code and assumptions
- Make compromises to get useful checking
 - Use simplifying assumptions to improve efficiency
 - Use heuristics to analyze common loop idioms
 - Accept some false positives and false negatives (unsound and incomplete analysis)

16 August 2001
David Larochelle

6

Implementation

- Extended LCLint
 - Open source checking tool [FSE '94] [PLDI '96]
 - Uses annotations
 - Detects null dereferences, memory leaks, etc.
- Integrated to take advantage of existing checking and annotations (e.g., modifies)
- Added new annotations and checking for buffer sizes

16 August 2001
David Larochelle

7

Annotations

- requires, ensures
- maxSet
 - highest index that can be safely written to
- maxRead
 - highest index that can be safely read
- char buffer[100];
 - ensures maxSet(buffer) == 99

16 August 2001
David Larochelle

8

SecurityFocus.com Example

```
char *strncat (char *s1, char *s2, size_t n)
/*@requires maxSet(s1)
    >=maxRead(s1) + n@*/
void func(char *str){
    char buffer[256];
    strncat(buffer, str, sizeof(buffer) - 1);
    return;
}
    uninitialized array
```

Source: Secure Programming working document,
SecurityFocus.com

16 August 2001
David Larochelle

9

Warning Reported

```
char * strncat (char *s1, char *s2, size_t n)
/*@requires maxSet(s1) >= maxRead(s1) + n @*/
char buffer[256];
strncat(buffer, str, sizeof(buffer) - 1);
```

strncat.c:4:21: Possible out-of-bounds store:
strncat(buffer, str, sizeof((buffer)) - 1);
Unable to resolve constraint:
requires maxRead (buffer @ strncat.c:4:29) <= 0
needed to satisfy precondition:
requires maxSet (buffer @ strncat.c:4:29)
 >= maxRead (buffer @ strncat.c:4:29) + 255
derived from strncat precondition:
requires maxSet (<parameter 1>
 >= maxRead (<parameter1>) + <parameter 3>

16 August 2001
David Larochelle

10

Overview of checking

- Intraprocedural
 - But use annotations on called procedures and global variables to check calls, entry, exit points
- Expressions generate constraints
 - C semantics, annotations
- Axiomatic semantics propagates constraints
- Simplifying rules
(e.g. $\text{maxRead}(\text{str}+i) \implies \text{maxRead}(\text{str}) - i$)
- Produce warnings for unresolved constraints

16 August 2001
David Larochelle

11

Loop Heuristics

- Recognize common loop idioms
- Use heuristics to guess number of iterations
- Analyze first and last iterations

Example:

```
for (init; *buf; buf++)
    – Assume maxRead(buf) iterations
    – Model first and last iterations
```

16 August 2001
David Larochelle

12

Case studies

- wu-ftpd 2.5 and BIND 8.2.2p7
 - Detected known buffer overflows
 - Unknown buffer overflows exploitable with write access to config files
- Performance
 - wu-ftpd: 7 seconds/ 20,000 lines of code
 - BIND: 33 seconds / 40,000 lines
 - Athlon 1200 MHz

16 August 2001
David Larochelle

13

Results

	instances in wu-ftpd (grep)	LCInt warnings with no annotations added	LCInt warning with annotations
strcat	27	19	12
strcpy	97	40	21
strncpy	55	4	4
Other Warnings	-	132 writes 220 reads	95 writes 166 reads

16 August 2001
David Larochelle

14

wu-ftpd vulnerability

```
int access_ok( int msgcode) {
    char class[1024], msgfile[200];
    int limit;

    ...

    limit = acl_getlimit(class, msgfile);
}
```



16 August 2001
David Larochelle

15

Related Work

- Lexical analysis
 - grep, its4, RATS, FlawFinder
- Wagner, Foster, Brewer [NDSS '00]
 - Integer range constraints
 - Flow insensitive analysis
- Dor, Rodeh and Sagiv [SAS '01]
 - Source-to-source transformation with asserts and additional variables.

16 August 2001
David Larochelle

16

Impediments to wide spread adoption

- People are lazy
- Programmers are especially lazy
- Adding annotations is too much work (except for security weenies)
- Working on techniques for automating the annotation process

16 August 2001
David Larochelle

17

Conclusion

- 2014:???
- Will buffer overflows still be common?
- Codify security knowledge in tools real programmers can use

Beta version now available:

<http://lclint.cs.virginia.edu>

David Larochelle

larochelle@cs.virginia.edu

David Evans

evans@cs.virginia.edu

16 August 2001
David Larochelle

18