



An Empirical Study on Detecting and Fixing Buffer Overflow Bugs

Linzhang Wang

Joint work with Tao Ye , Xuandong Li, Nanjing University, China

Lingming Zhang, University of Texas at Dallas, USA

June 6, 2016



软件新技术与产业化协同创新中心

Collaborative Innovation Center of Novel Software Technology and Industrialization



Outline



- **Background and motivation**
- Empirical study
- Experimental results
- Take home messages



Background and motivation



- Careless programming with unmanaged C/C++ languages may result in security vulnerabilities
 - Inappropriate memory manipulation
 - Mistaken assumptions about size
 - Makeup of a piece of data
 - Misuse of API
- **Buffer overflow** is one of the best known security vulnerabilities.
 - Missing Input validation or bound checking before memory manipulation or API calling may overwrite the allocated bounds of buffers.



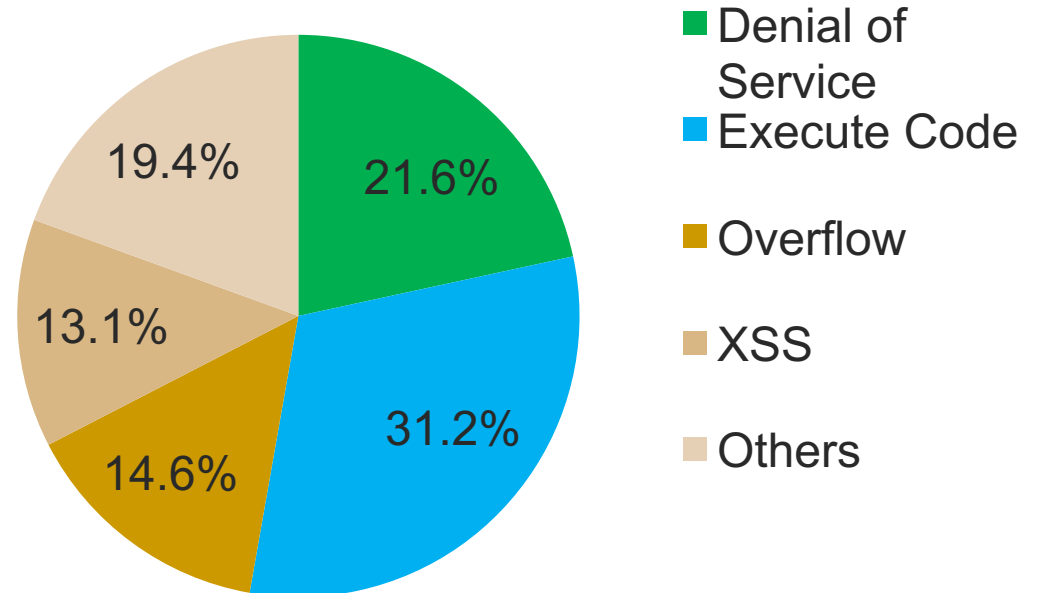
Background and motivation



■ Statistics of buffer overflows in CVE

- 14.6% of all, **3rd most** popular
- Prevalent attacks against legacy or newly deployed systems

Vulnerabilities by type



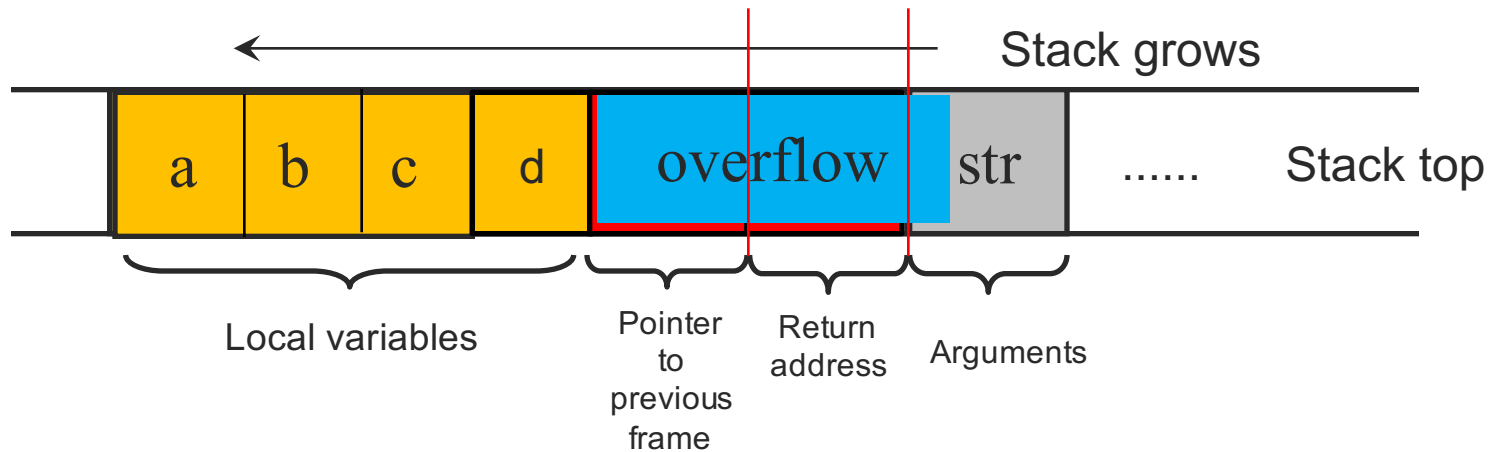


Background and motivation



■ Buffer overflow

```
void func (char *str) {  
    char buf[4];  
    strcpy(buf, str);  
}
```



■ str="abc"

■ str="abcdef..."



Background and motivation



- Buffer overflow causes severe damage
 - system crash
 - endless loop
 - executing arbitrary code

- What can we do to deal with buffer overflow?
 - Detection
 - Repair
 - Prevention/mitigation



Background and motivation



- Dynamic testing versus static analysis
 - Dynamic approach (Stackguard[1], CCured[2]):
 - Inserting special code into software to monitor buffer status
 - Advantage: few false-positives
 - Disadvantage: performance overhead, false-negatives
 - Static approach (Fortify, Checkmarx, Splint):
 - scanning source code
 - Advantage: discovering buffer overflow before software deployment, highly automated and scalable
 - Disadvantage: many false-positives



[1] C. Cowan, C. Pu, D. Maier, J. Walpole, P. Bakke, S. Beattie, A. Grier, P. Wagle, Q. Zhang, and H. Hinton, "Stackguard: Automatic adaptive detection and prevention of buffer-overflow attacks." in *USENIX Security*, vol. 98, 1998, pp. 63–78.

[2] G. C. Necula, S. McPeak, and W. Weimer, "Ccured: Type-safe retrofitting of legacy code," in *ACM SIGPLAN Notices*, vol. 37, no. 1, 2002, pp. 128–139.



Background and motivation



- Static techniques are widely used
- Few studies on effectiveness and efficiency of static techniques (Kratkiewicz[1])
- Conducting a study on static techniques
 - Effectiveness and efficiency of detecting buffer overflow

[1] K. J. Kratkiewicz, “Evaluating static analysis tools for detecting buffer overflows in c code”, Master’s thesis, Harvard University, 2005



Outline



- Background and motivation
- **Empirical study**
- Experimental results
- Take home messages



Empirical Study



- Research questions
- Subject system
- Selected techniques
- Experimental setup
- Experimental steps



Research questions

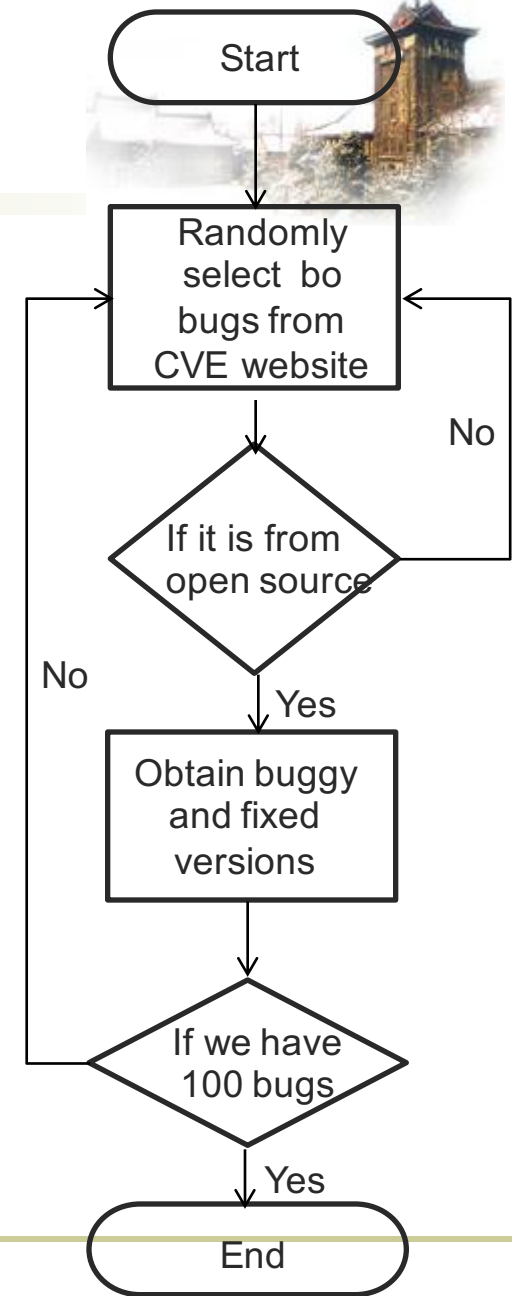


- RQ1. Effectiveness?
 - False positive and false negative
- RQ2. Efficiency?
 - Resource consumption
- RQ3. API?
 - Root cause of buffer overflow vulnerabilities
- RQ4. Manual fix patterns?
 - Official repair



Subject systems

- Selection criterion
 - Open-source
 - Buffer overflow
- 100 buffer overflow bugs from 63 real-world projects, totaling 28MLoc, ranging from CVE-1999 to CVE-2014





Subject systems



Subject	Description	Size(LoC)	# BO bugs (#Versions)
mapserver	Platform for publishing data to web	276K	4(2)
libzip	Library for handling zip archives	10K	1(1)
man	Command used to display user manual	10K	3(2)
libthai	Thai language support routines	6K	1(1)
...
Total		28M	100(81)



Static techniques




- Representative static techniques: Fortify, Checkmarx, and Splint
 - According to Gartner Group report, Fortify and Checkmarx are leading commercial products in application security market.
 - Splint is one of the first open-source tools that concern safety issues, and it is widely used.



Static techniques



- Fortify (version 5.10.1.0043) 
- Code compiled
- Data flow analysis – source and sink
- Control flow analysis -- a set of operations
- Semantic analysis – dangerous use of functions and APIs

- <http://www8.hp.com/us/en/software-solutions/application-security/index.html>



Static techniques



■ Checkmarx (version 7.1.6)



- Un-compiled, incomplete code
- Meticulous model between users and other data
- Tracking data and logic flows
- Identifying vulnerabilities according to static analysis rules
- <https://www.checkmarx.com>



Static techniques



- Splint (version 3.1.1)
 - Modeling buffer and annotating buffer size
 - Precondition and postcondition of buffer access
 - Constraint solver
 - <http://www.splint.org>



Experimental setup



- Fortify and Splint:
 - on a server with Intel Xeon CPU E5-2603 (1.80GHz) and 128GB RAM on Ubuntu Linux 12.04

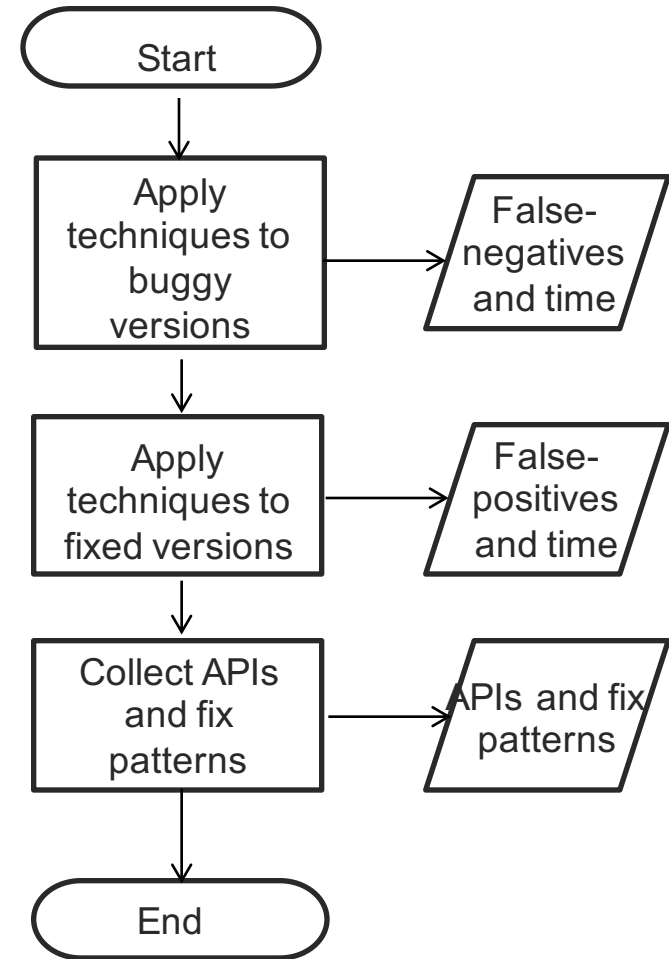
- Checkmarx
 - on a server with Intel Xeon CPU E5-2650 (2.30GHz) and 384GB RAM on Windows Server 2008



Experimental steps



- Apply all techniques to the subjects
 - buggy version-> find the bugs that cannot be detected ->false-negatives.
 - fixed version->find the fixed bugs that are still identified as bugs->false-positives.
- Categorize the root cause of BO->API
- Categorize the manual fix pattern.





An example



GCC(1)

GNU

GCC(1)

NAME

gcc - GNU project C and C++ compiler

SYNOPSIS

```
gcc [-c|-S|-E] [-std=standard]  
    [-g] [-pg] [-Olevel]  
    [-Wwarn...] [-Wpedantic]  
    [-Idir...] [-Ldir...]  
    [-Dmacro[=defn]...] [-Umacro]  
    [-foption...] [-mmachine-option...]  
    [-o outfile] [@file] infile...
```

Only the most useful options are listed here; see below for the remainder. `g++` accepts mostly the same options as `gcc`.

DESCRIPTION

When you invoke GCC, it normally does preprocessing, compilation, assembly and linking. The "overall options" allow you to stop this process at an intermediate stage. For example, the `-c` option says not to run the linker. Then the output consists of object files output by the assembler.

Manual page gcc(1) line 1 (press h for help or q to quit)



CVE information



- ID: CVE-2001-1028
- Description: Buffer overflow in ultimate_source function of man 1.5 and earlier allows local users to gain privileges.



Source code



Buggy code (man-1.5i2):

```
286 static const char*
287 ultimate_source(const char*
    name0) {
...
297 static char
    ultname[BUFSIZE];
298
299 strcpy(ultname, name0);
...
}
```

Fixed code (man-1.5p):

```
291 static const char*
292 ultimate_source(const
    char* name0) {
...
302 static char
    ultname[BUFSIZE];
303
304 if (strlen(name0) >=
    sizeof(ultname))
305 return name0;
306 strcpy(ultname, name0);
...
}
```



Results



- On the buggy version
 - Fortify: detected, 3m50s
 - Checkmarx: detected, 2m25s
 - Splint: preprocess error
- On the fixed version
 - Fortify: undetected (fixed), 3m58s
 - Checkmarx: detected (not fixed), 2m25s
 - Splint: preprocess error
- API: strcpy
- Fix pattern: add boundary check



Outline



- Background and motivation
- Empirical study
- **Experimental results**
- Take home messages



RQ1: Effectiveness



Techs	# Identified Bugs	FN Rate	# Identified Fixes	FP Rate
-------	-------------------	---------	--------------------	---------

By combining these techniques, we can get a lower false-negative rate.

The cost is a relatively higher false-positive rate.

Checkmarx can detect most buffer overflow bugs.

Splint performs best in terms of false-negative rate.

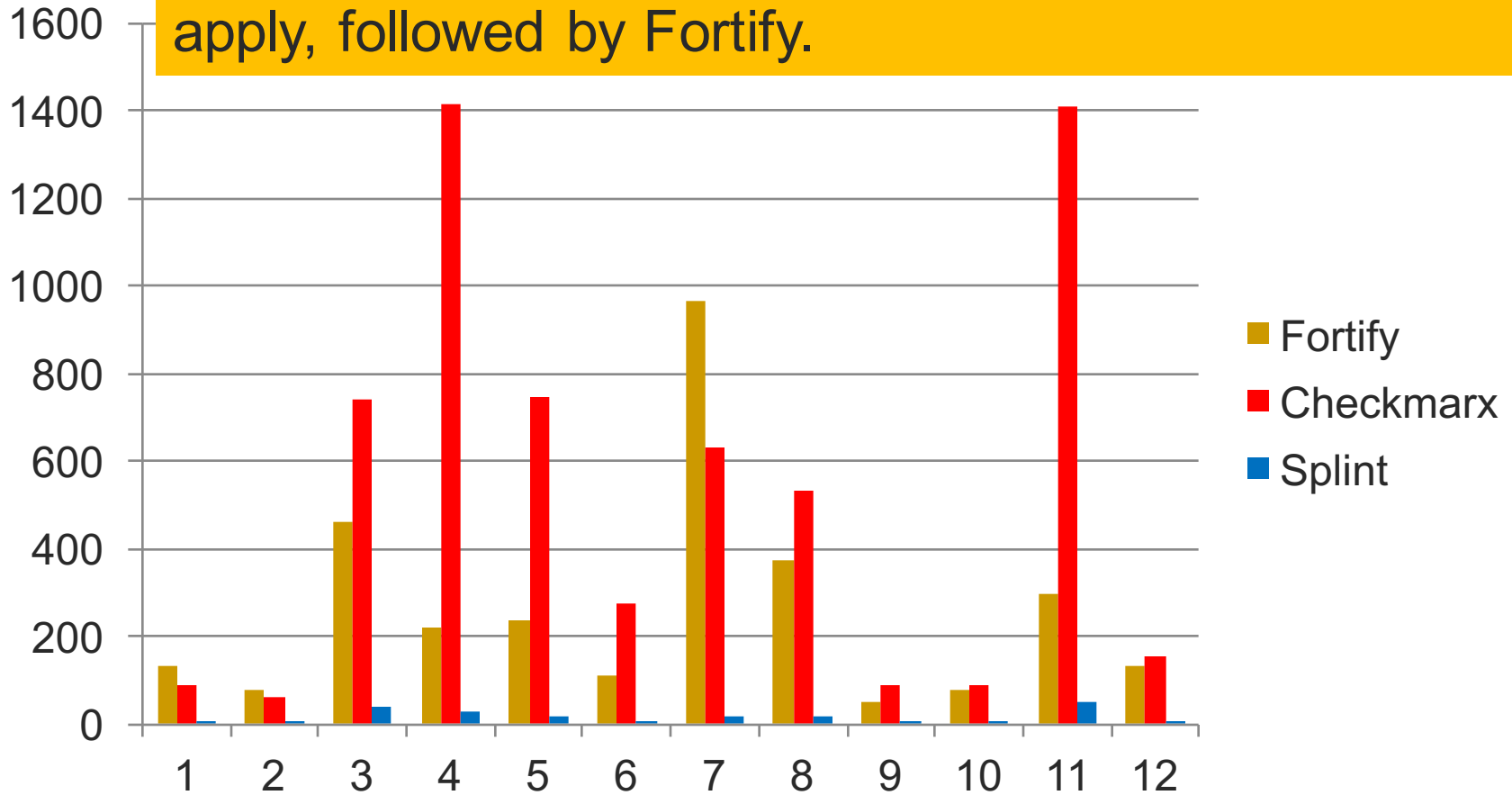
Fortify performs best in terms of false-positive rate.



RQ2: Efficiency



(s) Checkmarx tends to be the most costly technique to apply, followed by Fortify.

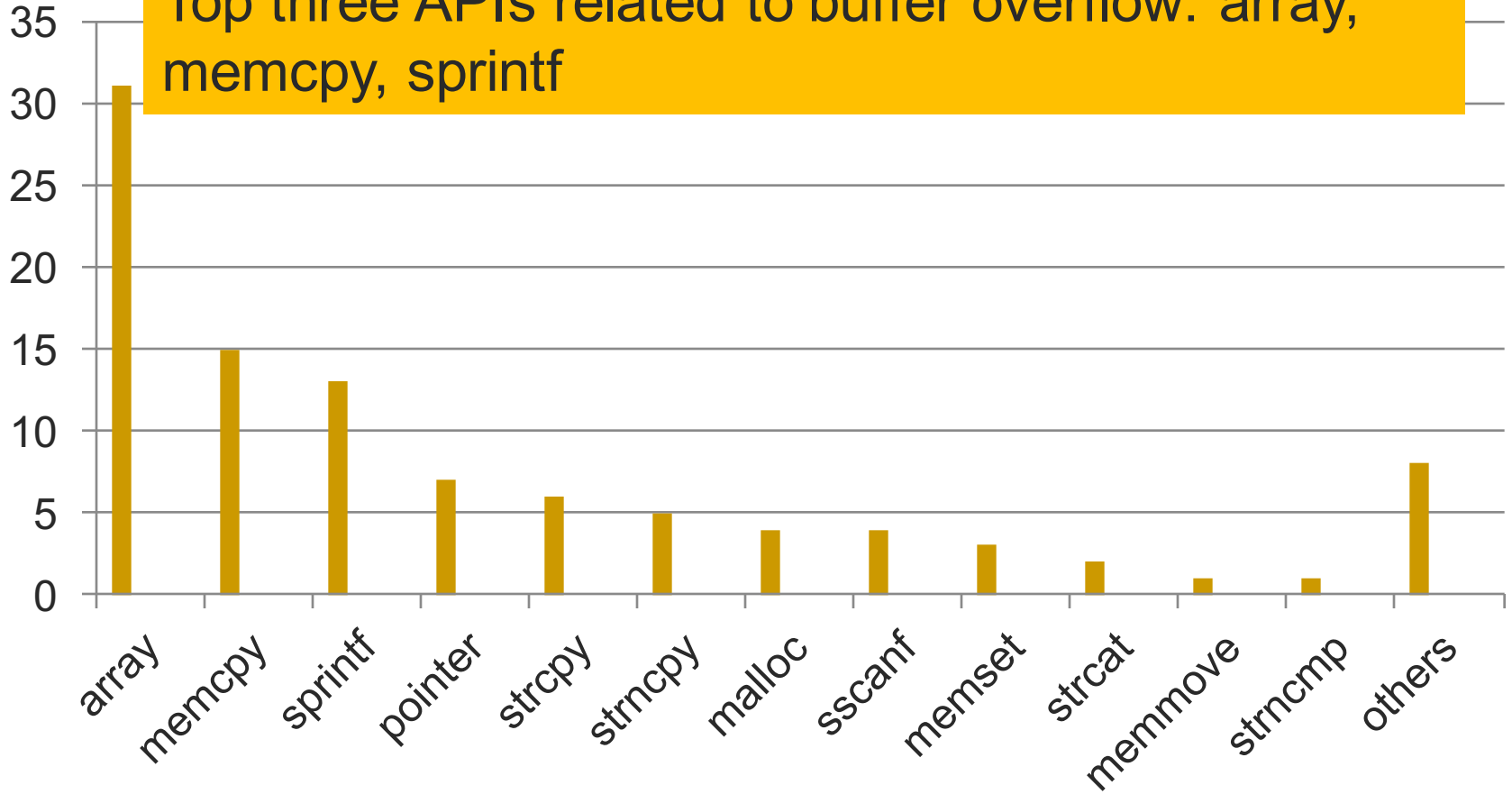




RQ3: API distribution



Top three APIs related to buffer overflow: array, memcpy, sprintf





APIs and techniques



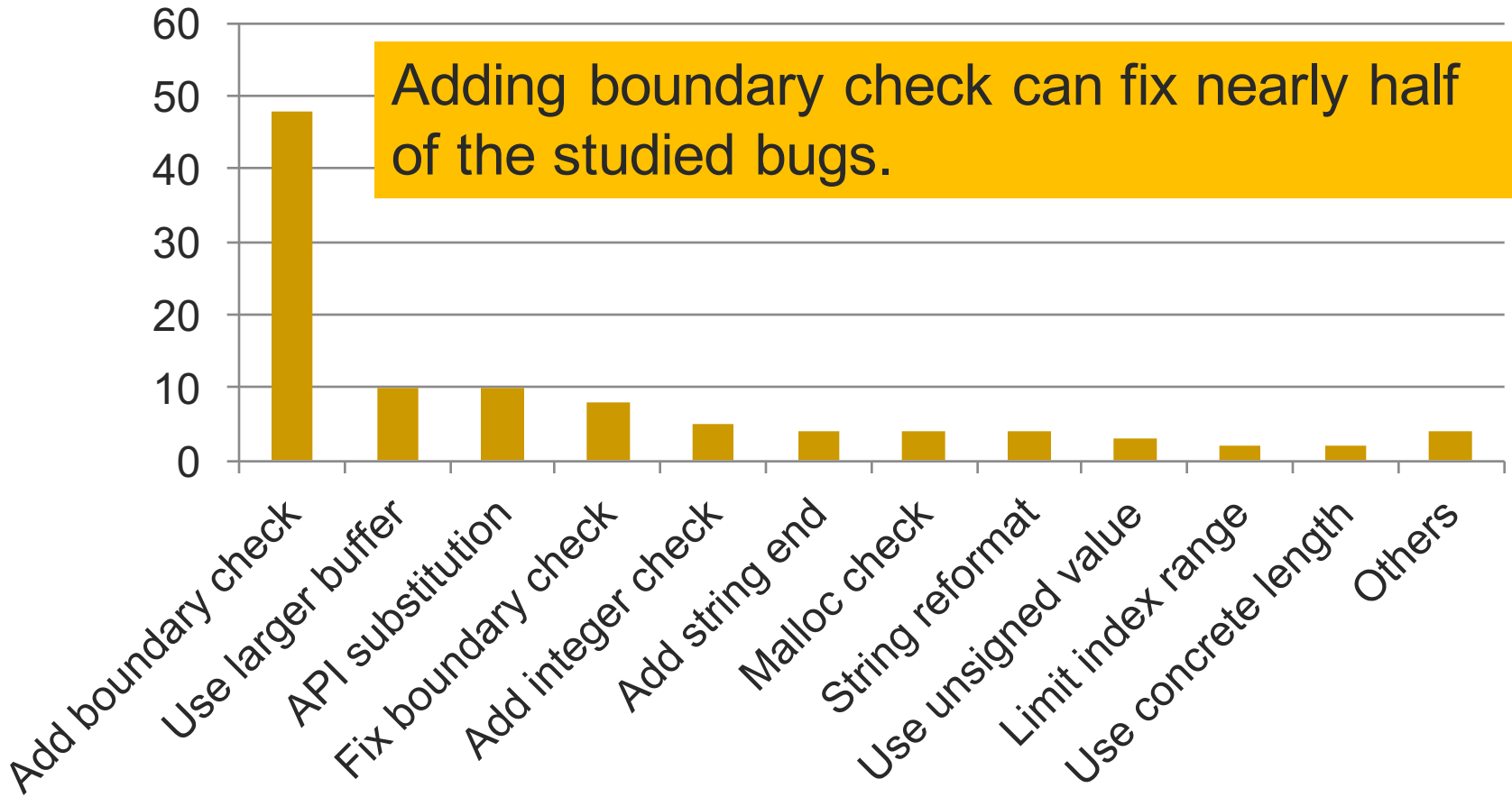
API	# Instances	Fortify	Checkmarx	Splint
array	31	4/21	2/31	8/12
memcpy	15	0/5	9/15	0/1
sprintf	13	Splint works well on array.		
pointer	7	Splint reports a warning when the constraints are unsolvable.		
strcpy	6	Evidence: Splint identifies 0/8 fixes		
strncpy	5			
...		...		

Fortify and Checkmarx can find most bugs on APIs like sprintf and strcpy. They tend to report a warning on unsafe API.

Evidence: strncpy 1/4 1/5



RQ4: Fix strategy distribution





Fix strategy and API



- Most APIs prefer ‘add boundary check’
- For some of them, there may be a more suitable way.

Strategies	array	memcpy	sprintf	...	Total
Add boundary check	18	9	2	...	48
Use larger buffer	3	1	4	...	10
API substitution	-	1	6	...	10
...	
Total	31	15	13	...	



Outline



- Background and motivation
- Empirical study
- Experimental results
- **Take home messages**



Take home messages



- Effectiveness:
 - Use Fortify alone to achieve low false-positive rate.
 - Use techniques together to achieve low false-negative rate.
- Efficiency:
 - Checkmarx is the most costly technique among studied techniques.



Take home messages



■ API:

- Array, memcpy and sprintf are top three APIs related to buffer overflow.
- Splint has the lowest false-negative rate on array
- Fortify can find most bugs on APIs like sprintf and strcpy, followed by Checkmarx.



Take home messages



- Fix strategy:
 - Most buffer overflow bugs can be fixed using strategy ‘adding boundary check’.
 - If ‘adding boundary check’ fails, one can choose other strategy according to the API involved.



Conclusions



- A quantitative study of the state-of-art static techniques for buffer overflow detection on 100 bugs from 63 real-world projects totaling 28 MLoC
- A qualitative analysis of the false-positives and false-negatives of studied static detection techniques, which can guide the design and implementation of more advanced buffer overflow detection techniques.
- A categorization on the fix patterns of buffer overflow bugs to guide both manual and automated buffer overflow repair techniques.



Ongoing and Future Work



- Automatic static Buffer Overflow Warning Inspection
 - Manual inspection of static report is time consuming and lab intensive
 - Static warning + dynamic symbolic execution
- Automatic Buffer Overflow Bug Repair
 - Manual repair needs programming expertise and may introduce new bugs.
 - For validated true buffer overflow vulnerabilities, we automatically generate fix suggestions according to the predefined templates which are created based on human repair patterns.



■ More info

- Tao Ye, Lingming Zhang, Linzhang Wang and Xuandong Li. An Empirical Study on Detecting and Fixing Buffer Overflow Bugs, in proceedings of ICST2016, Aril 9-16, 2016, Chicago, US.
- <http://bo-study.github.io/Buffer-Overflow-Cases/>

■ Contact

- lzwang@nju.edu.cn



■ Thanks

■ Questions?