ISC 2017

2017 中国互联网安全大会
China Internet Security Conference

Towards Efficient Heap Overflow Discovery

Chao Zhang
Assoc. Prof.
Tsinghua University

About Me

Experience

- Tsinghua University, Assoc. Prof., 2016/11-present
- UC Berkeley, Postdoc, 2013/9-2016/9, Advisor: Dawn Song
- Peking University, Ph.D., 2008/9-2013/7, Advisors: 邹维，韦韬
- Peking University, Undergraduate, 2004/9-2008/7, Math

Honors

- Young Elite Scientists Sponsorship Program by CAST
- DARPA CGC, Captain of Team CodeJitsu
  - Defense #1 in 2015 CQE, Attack #2 in 2016 CFE
- Microsoft BlueHat Prize Contest 2012
  - Special Recognition Award
- DEFCON CTF 2015 (#5), 2016 (#2), 2017 (#5)
- GeekPwn 2017/5/12

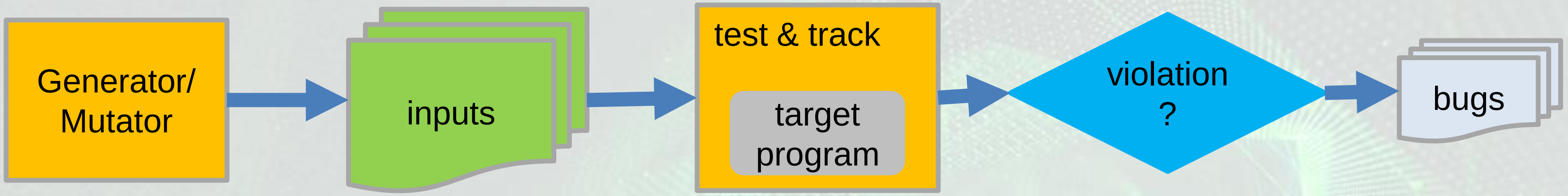
Agenda

- Basics of Fuzzing
- Studies on Fuzzing
- HOTracer: Offline Dynamic Solution
- Conclusion

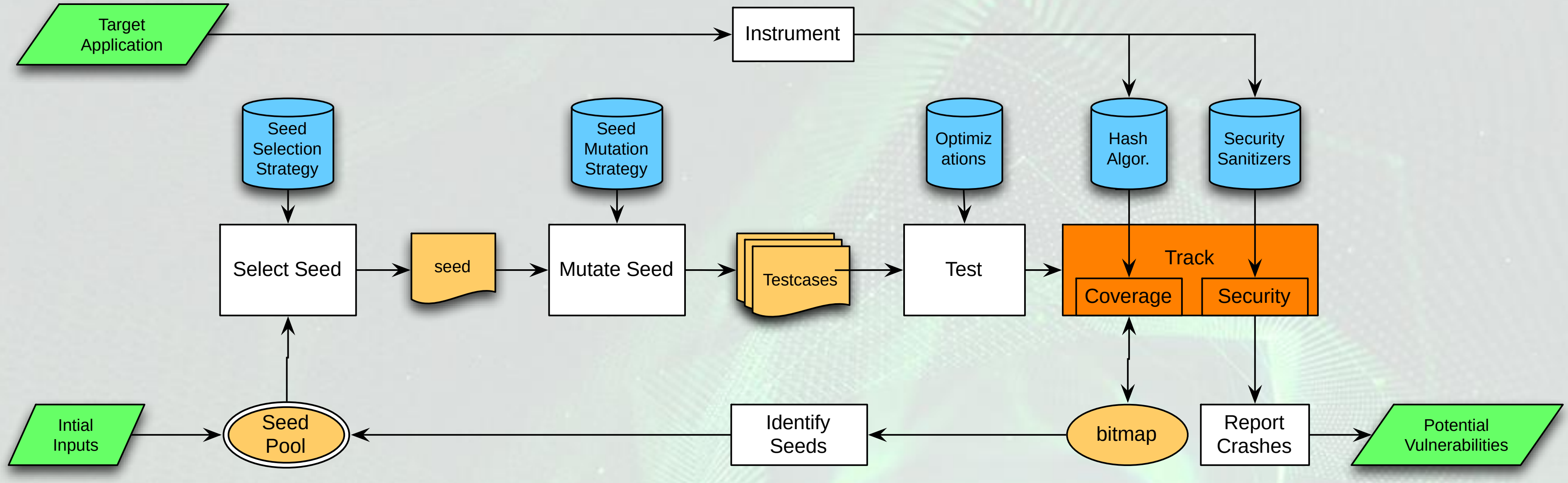
Vulnerability Detection Solutions

- Static Analysis
- Taint Analysis
- Fuzzing
  - mutation, generation
  - blackbox, greybox, whitebox
  - smart, dumb
- Symbolic Execution
- Dynamic Detection
  - online
  - offline

Basics of Fuzzing



Basics of Fuzzing: AFL



- Secret ingredient of AFL:
  - Throughput
  - Code Coverage
  - Exception Capturing

Key Questions of Fuzzing

- How to get initial inputs?
- How to select seed from the pool?
- How to generate new testcases?
  - How to mutate seeds? Location and value.
- How to efficiently test target application?
- How to track the testing?
  - Code coverage, Security violation, ...?
- How do we update the seed pool?
  - identify good testcases, shrink seed pool...

How to get initial inputs?

- Why is it important?
  - cpu time
  - complex data structure
  - hard-to-reach code
  - reusable between fuzzings
- Solutions
  - standard benchmarks
  - crawling from the Internet
- Extra step
  - distill the corpus

How to select seed from the pool?

- Why is it important?
  - prioritize seeds which are more helpful,
    - e.g., cover more code, more likely to trigger vulnerabilities
  - save computing resources
  - faster to identify hidden vulnerabilities
- Solutions
  - AFLFast (CCS’ 16): seeds being picked fewer or exercising less-frequent paths
  - Vuzzer (NDSS’ 17): seeds exercising deeper paths
  - QTEP (FSE’ 17): seeds covering more faulty code
  - AFLgo (CCS’ 17): seeds closer to target vulnerable paths
  - SlowFuzz (CCS’ 17): seeds consuming more resources

How to generate new testcases?

- Why is it important?
  - explore more code in a shorter time
  - target potential vulnerable locations
- Solutions
  - Vuzzer (NDSS’ 17):
    - where to mutate: bytes related to branches
    - what value to use: tokens used in the code.
  - Skyfire (Oakland’ 17):
    - learn Probabilistic Context-Sensitive Grammar from crawled inputs
  - Learn&Fuzz (Microsoft):
    - learn RNN from valid inputs

How to track the testing?

- Why is it important?
  - Code coverage: leading to thorough program states exploring
  - Security violations: capturing bugs that have no explicit results
- Solutions
  - Code coverage:
    - AFL bitmap, SanitizerCoverage
  - Security violations:
    - AddressSanitizer
    - UBSan
    - MemorySanitizer
    - ThreadSanitizer
    - DataFlowsanitizer
    - LeakSanitizer
    - ...

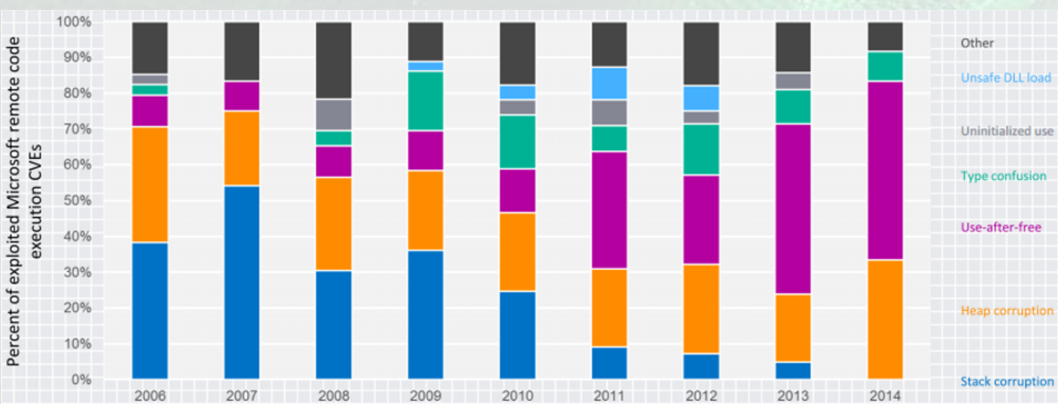
Agenda

- Basics of Fuzzing
- Studies on Fuzzing
- HOTracer: Offline Dynamic Solution
- Conclusion

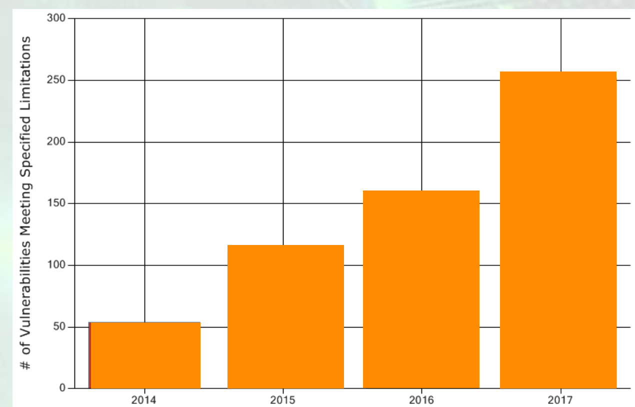
Motivation: Heap Overflow

- Stack overflow exploits are rare
  - Defenses: ASLR, shadow stack, Stackguard, StackArmor, etc.
  - Compiler efforts

- Heap overflows become dominant
  - exploit techniques become reliable: heap spray, heap fengshui, etc.



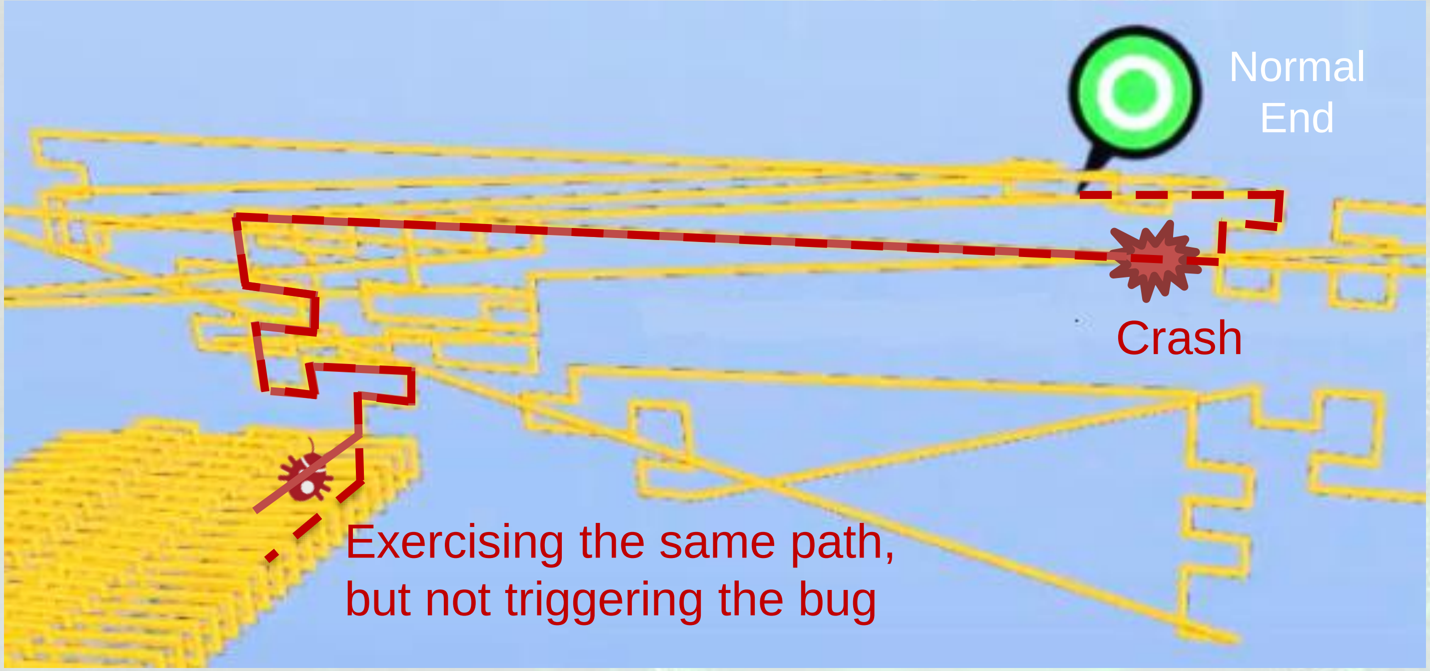
Microsoft RCE exploitation trend (2008-2014)



Heap overflow vulnerabilities in NVD (2014-2017)

Why Fuzzing Fails?

- Value sensitive, not only path sensitive



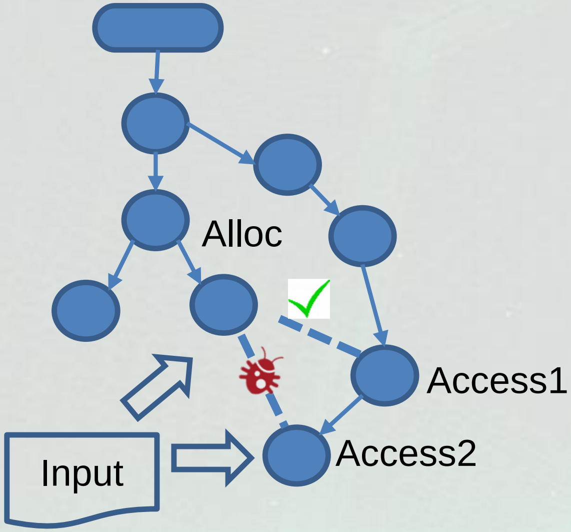
Root cause : Spatial inconsistency

- Example of a heap overflow
  - Heap allocation operations
    (obj, $size_{allocation}$)
  - Heap access operations
    (ptr, $offset_{ptr}$, $size_{access}$)
- Heap overflow happens
  - When input size = SIZE

```
#define SIZE (1024-4)
struct OBJ{
  char name[SIZE];
  void set_name(char* src, size_t size){
    if(size > SIZE) exit(-2);
    memcpy(name, src, size);
    // off-by-one, when size == SIZE
    name[size]=0;
  }
};
int main(){
  OBJ* p1 = new OBJ();
  OBJ* p2 = new OBJ();
  // tainted: size and input
  input = get_input(&size);
  // Vul #1: off-by-one if size=SIZE
  p1->set_name(input, size);
  // coalesce p1 and p2, causing p1 free.
  free(p2);
  // Vul #2: use after free
  printf("p1 name: %s\n", p1->name);
  return 0;
}
```

Heap overflow condition: $\text{Range}_{access} > \text{Range}_{obj}$

Root Cause: Controllability (Taint)

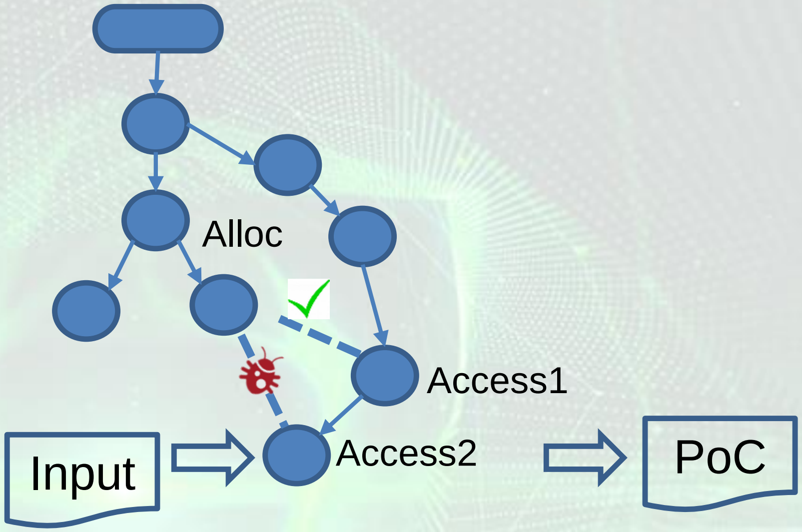


An Example Call Graph

	Alloc controllable	Access controllable	How to overflow
S1	N	N	inherent errors (N/A)
S2	N	Y	Access out of bound
S3	Y	N	Allocate small buffer
S4	Y (bytes_1)	Y (bytes_2)	Change the value of two sides independently
S5	Y (bytes_0)	Y (bytes_0)	Check IO2BO (e.g., (x+2, x+1))

Our solution

- Get execution traces
- Identify heap operations
  - group into pairs <alloc, access>
- Track spatial attributes
- Track taint attributes
- Build vulnerability conditions
- Generate PoC inputs



Xiangkun Jia, **Chao Zhang**, Purui Su, Yi Yang, Huafeng Huang, Dengguo Feng, Towards Efficient Heap Overflow Discovery, USENIX Security 2017

Implementation Choices

- Get execution traces
- Identify heap operations
  - group into pairs <alloc, access>
- Track spatial attributes
- Track taint attributes
- Build vulnerability conditions
- Generate PoC inputs

- representative inputs
- custom heap managers
  - lineage analysis
- fine-grained taint tracking
- symbolic execution
- spatial inconsistency
- constraint optimization & solving

Bug Finding Results

ID (count)	Application	version	input	bug status
new (1)	Feiq	3.0.0.2	tcp	reported
new (1)	WMPlayer	12.0.7601	mp4	reported
new (3)	VLC	2.2.1	mp4	fixed
new (1)	VLC	2.2.4	mp4	reported
new (2)	iTunes	12.4.3.1	mp4	reviewing
new (1)	ffmpeg	c0cb53c	mp4	CVE
new (6)	QQPlayer	3.9(936)	mp4	rewarded
new (1)	QQMusic	11.5	m4a	rewarded
new (1)	BaiduPlayer	5.2.1.3	mp4	reviewing
new (2)	RealPlayer	16.0.6.2	mp4	CVE
new (1)	MPlayer	r37802	mp4	reported
new (3)	KMPlayer	3.9.1.138	mp4	fixed
new (4)	KMPlayer	4.1.1.5	mp4	reported
new (7)	Potplayer	1.6.60136	mp4	fixed
new (2)	Potplayer	1.6.62949	mp4	reported
new (5)	Splayer	3.7	mp4	reported
new (2)	MS Word	2007,10,16	rtf	reviewing
new (1)	WPS Word	10.1.0.5803	doc	reported
new (2)	OpenOffice	4.1.2	doc	reviewing
new (1)	IrfanView	4.41	m3u	fixed

47 vulnerabilities in 17 applications

Case Studies

- Tainted access offset
  - offset is influenced by input

```
Dst = Object + tainted_offset
memcpy(Dst, Src)
```

- Implicit taint:
  - Allocation size is based on the input length
  - Access size implicitly depends on input

```
while (input != '\0' ) length++;
buffer = malloc(length);
```

```
while(input == val){
        memcpy();
}
```

- Multiple vulnerabilities in one trace
  - Two extra vulnerable points in the same trace as CVE-2014-1761
- Long testing time
  - A VLC vulnerability occurred only after several minutes

Conclusions

- Fuzzing is still the most popular vulnerability discovery solution.
- Fuzzers could be improved in several ways.
- We point out that fuzzers may miss heap overflow vulnerabilities (and other type of vulnerabilities) due to its value insensitivity.
- We propose a new solution HOTracer, able to identify potential heap overflows in a given trace.
- We found 47 vulnerabilities in 17 real world applications, including 2 vulnerable points in Microsoft Word.

THANKS

中国互联网安全大会

360互联网安全中心