Towards Efficient Heap Overflow Discovery

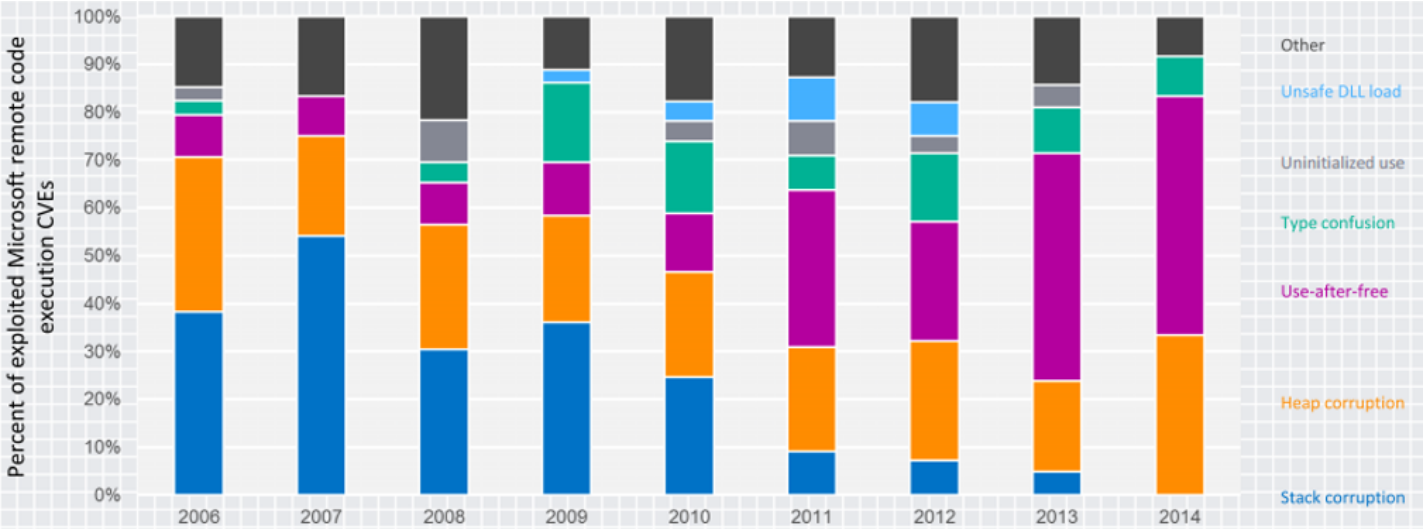
Xiangkun Jia, Chao Zhang, Purui Su,
Yi Yang, Huafeng Huang, Dengguo Feng

TCA/SKLCS lab, Institute of Software, Chinese Academy of Sciences,
University of Chinese Academy of Sciences,
Institute for Network Science and Cyberspace, Tsinghua University

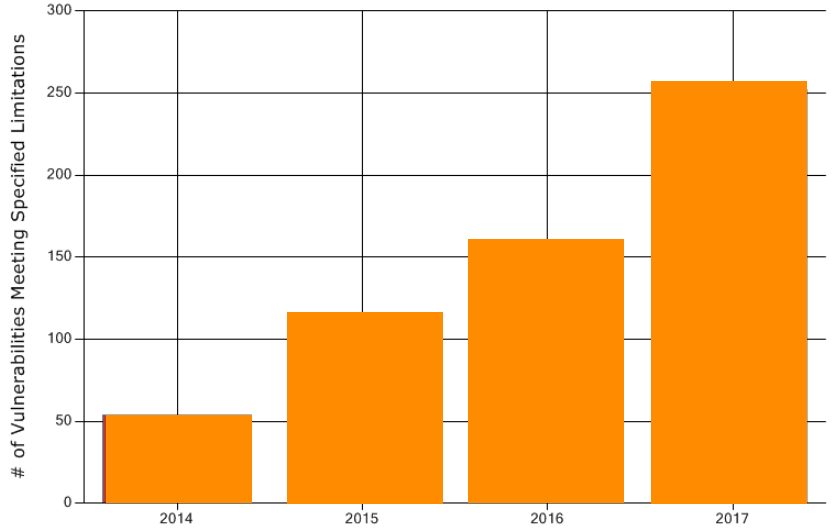
Contents

Motivation

- Stack overflow exploits are rare
  - Defenses are deployed, e.g., ASLR, shadow stack, Stackguard, StackArmor, etc.
- Heap overflows vulnerabilities become popular and realistic
  - Attack techniques are developing, e.g., dword shoot, heap spray, heap fengshui, etc.



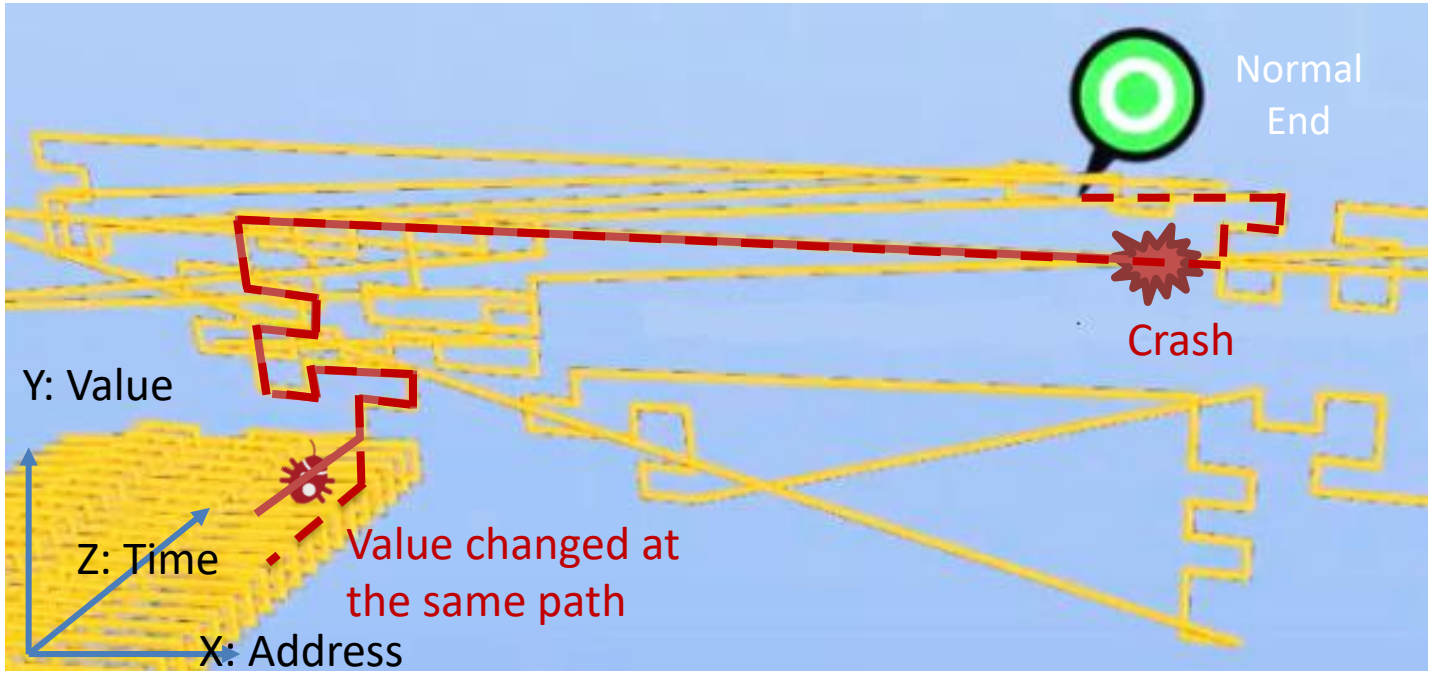
Microsoft RCE exploitation trend (2008-2014)



Heap overflow vulnerabilities in NVD (2014-2017)

Existing methods

- Runtime detections are passive
  - rely on triggering vuls
- Fuzzers are sometimes blind
  - just bypass vuls because of value
- Dynamic analysis is lack of a complete model
  - find heap overflow by integer overflow




```
X = input()
buffer1 = malloc(X+2)
//Heap overflow only if X+2 is integer overflow but X+1 not
memcpy(buffer1, “xxxx”, X+1)
```

Goal

- More proactive
  - Find vulnerabilities before attackers
- More targeted
  - Focus on heap overflow vulnerabilities
- More complete
  - Build a complete model of vulnerabilities

Root cause : Spatial inconsistency

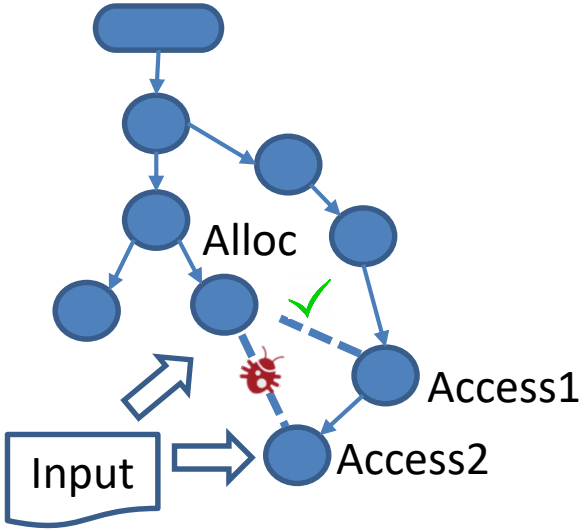
- Example of a heap overflow
  - Heap allocation operations (obj, $size_{allocation}$)
  - Heap access operations (ptr, $offset_{ptr}$, $size_{access}$)
- Heap overflow happens
  - When input size = SIZE

```
1  #define SIZE (1024-4)
2  struct OBJ{
3    char name[SIZE];
4    void set_name(char* src, size_t size){
5      if(size > SIZE) exit(-2);
6      memcpy(name, src, size);
7      // off-by-one, when size == SIZE
8      name[size]=0;
9    }
10 };
11 int main(){
12   OBJ* p1 = new OBJ();
13   OBJ* p2 = new OBJ();
14   // tainted: size and input
15   input = get_input(&size);
16   // Vul #1: off-by-one if size=SIZE
17   p1->set_name(input, size);
18   // coalesce p1 and p2, causing p1 free.
19   free(p2);
20   // Vul #2: use after free
21   printf("p1 name: %s\n", p1->name);
22   return 0;
23 }
```

Heap overflow condition: $Range_{access} > Range_{obj}$

Observation : Controllable attributes

- Types of heap overflow (based on controllable attributes)

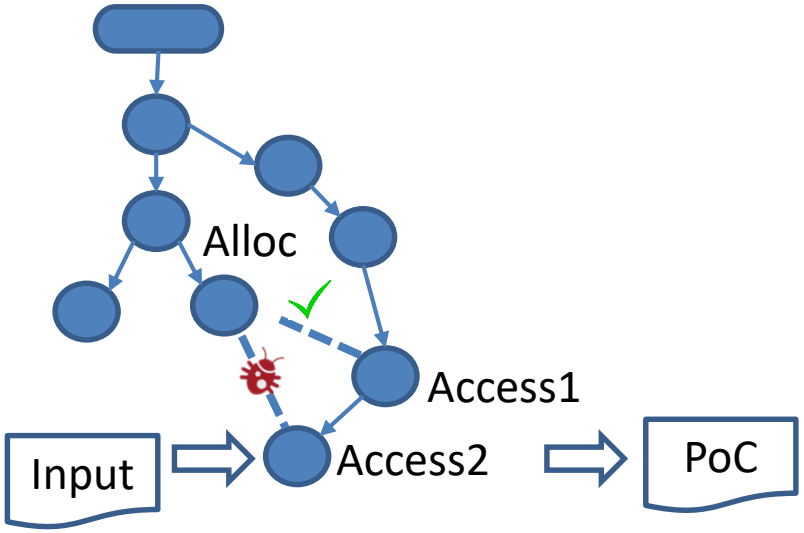


Call graph of an example

	Alloc controllable	Access controllable	How to overflow
S1	N	N	--/inherent errors
S2	N	Y	Copy more data/Copy to outside offset
S3	Y	N	Allocate small buffer/allocation size is integer overflow
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))

Overview of HOTracer

1. Get execution traces
2. Identify heap operations
3.1 Track spatial attributes
3.2 Track controllable(taint) attributes
4. Build vulnerability conditions
5. Generate PoC inputs



Challenges

Make it efficient !

1. Get execution traces	——	Get representative traces efficiently
2. Identify heap operations	——	Handle custom function implementations
3.1 Track spatial attributes	——	Pair heap operations accurately
3.2 Track controllable(taint) attributes	——	Use fine-grained taint analysis
4. Build vulnerability conditions	——	Based on the complete model
5. Generate PoC inputs	——	Consider path constraints

Step1. Get execution traces - **Get representative traces efficiently**

- Testcase selection
  - Diverse sources: mutator, crawler
  - Select representative ones: min-set coverage analysis based on sub-types (e.g., tags in MP4)

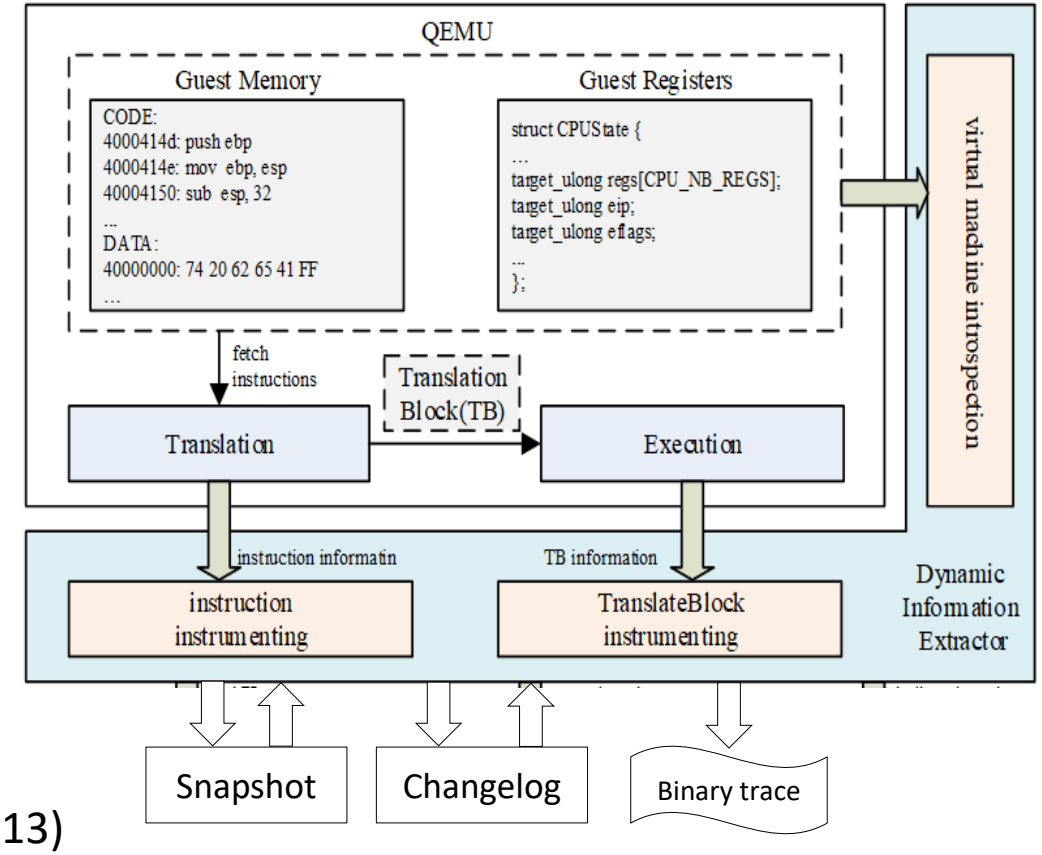
ftyp						√	file type and compatibility
pdin							progressive download information
moov						√	container for all the metadata
	mvhd					√	movie header, overall declarations
	trak					√	container for an individual track or stream
		tkhd				√	track header, overall information about the track
		tref					track reference container
		edts					edit list container

MP4 file structures

Implementation

Step1. Get execution traces - **Get representative traces efficiently**

- Binary instrument with record and replay

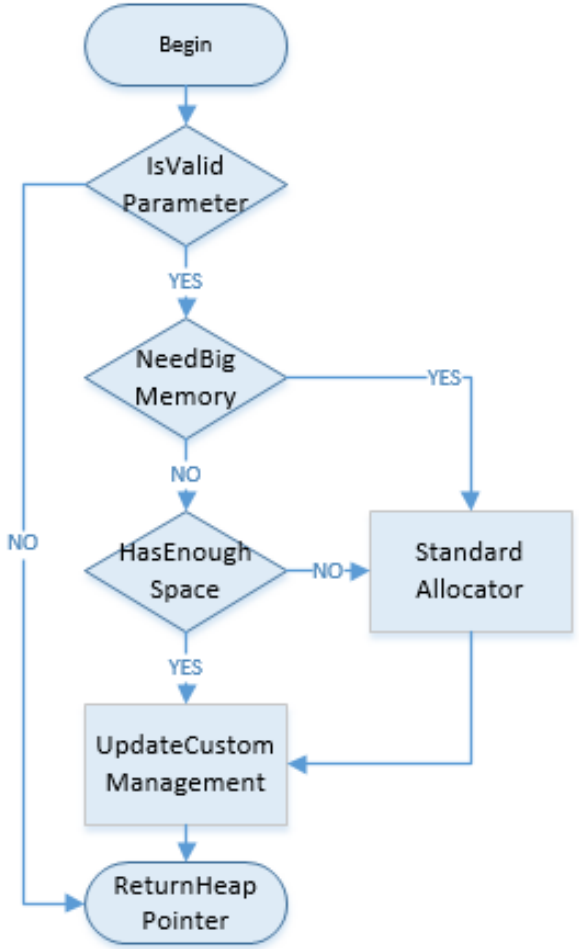


Our instrument:
Xede (RAID' 15)

Inspired by
PANDA (CCS' 13)

Step2. Identify heap operations - **Handle custom function implementations**

- Custom Heap allocation operations
  - Functionality
    - A. return a heap pointer
  - Process inside function
    - B1. invoke standard allocation interfaces
    - B2. response to allocation applications bases on different sizes
    - B3. pad extra bytes to align
    - B4. update heap management structures in a critical way
  - Used way
    - C1. return space is written before read
    - C2. allocator is called many times
    - C3. allocator returns different space
    - C4. allocator sometimes initials the space



High-level workflow of custom allocators

Step2. Identify heap operations - **Handle custom function implementations**

- Custom Heap access operations
  - Access the buffer in loops

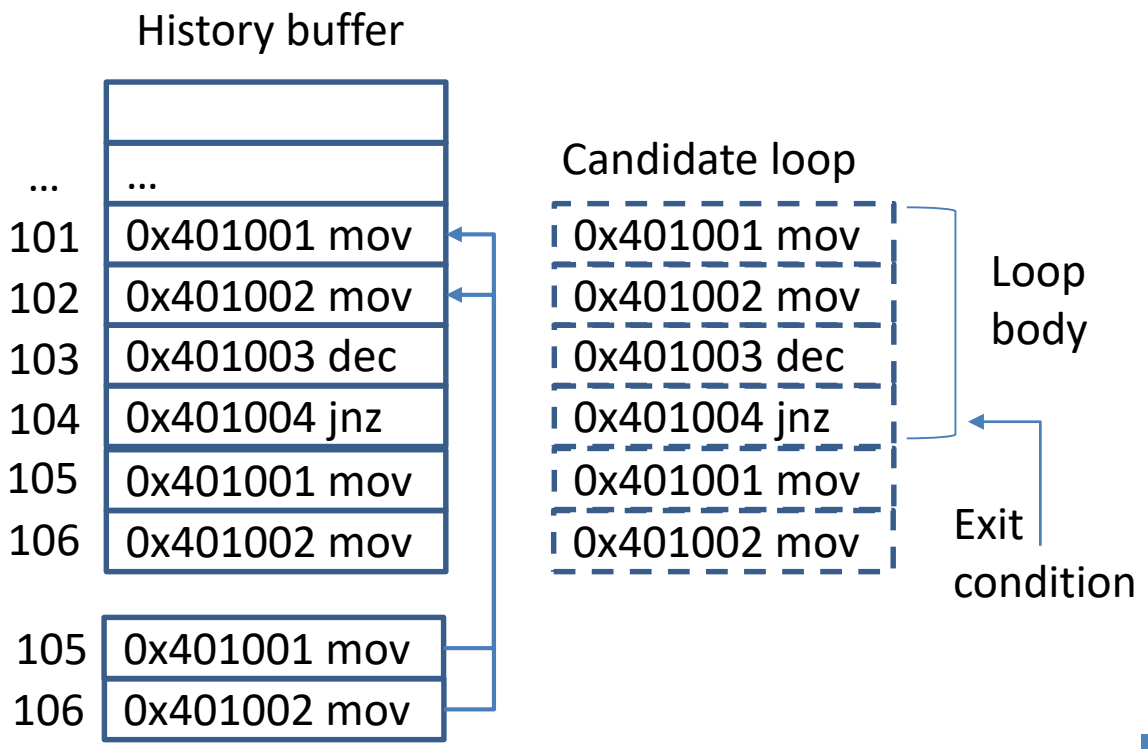
```
loc_781807A3:                                   ; CODE XREF: _strncpy+2A↑j
                                                ; _strncpy+64↓j
.text:781807A3    mov     al, [esi]
.text:781807A5    add     esi, 1
.text:781807A8    mov     [edi], al
.text:781807AA    add     edi, 1
.text:781807AD    test    al, al
.text:781807AF    jz      short loc_781807E8
.text:781807B1    sub     ebx, 1
.text:781807B4    jnz     short loc_781807A3
```

  - Access the buffer with REP instructions

```
.text:7855ACA0                  rep movsd
```

  - Access the buffer with continuous instructions

```
.text:7855ADF8 loc_7855ADF8:
.text:7855ADF8
.text:7855ADF8                  mov     al, [esi+3]
.text:7855ADFB                  mov     [edi+3], al
.text:7855ADFE                  mov     al, [esi+2]
.text:7855AE01                  mov     [edi+2], al
.text:7855AE04                  mov     eax, [ebp+8]
.text:7855AE07                  pop     esi
.text:7855AE08                  pop     edi
.text:7855AE09                  leave
.text:7855AE0A                  retn
```



Identify loops as continuously repeated sub-strings, ending with jump instructions

Step3. Track Spatial Attributes - **Pair heap operations accurately**

- Pair heap access operations to heap allocation operations
  - Pointer taint analysis
    - Taint source: heap pointers returned from allocators
    - Heap access: dereference of a heap pointer
  - Under-taint issue

```
//EAX is a pointer, EBX is another pointer
//EAX (Taint_EAX ) EBX (Taint_EBX )
SUB EAX, EBX
//previous solutions bleach EAX's taint
MOV ESI, EAX
…
//ESI=(EAX-EBX)+EBX=EAX, not related to EBX
ADD ESI, EBX
```

Previous solution	Signed taint label
→ EAX (NULL)	→ EAX ($Taint_{EAX}$, —$Taint_{EBX}$)
ESI (NULL)	ESI ($Taint_{EAX}$, —$Taint_{EBX}$)
…	…
→ ESI ($Taint_{EBX}$)	→ ESI ($Taint_{EAX}$)

Implementation

Step3. Track Taint Attributes - **Use fine-grained taint analysis**

- Multiple label taint analysis at byte level
- Kernel instructions: NOT record

- Sub-Step: Heap Operation Pairs Sorting
  - Select more vulnerable pairs
  - Filter heap management functions

Implementation

Step4. Build Vulnerability Conditions - **Based on the complete model**

- $Range_{access} > Range_{obj}$
- Consider integer overflow (IO2BO)

Step5. Generate PoC inputs - **Consider path constraints**

- Analyze from the last read to the vulnerable point
- Only symbolize relevant input bytes
- Use Z3 to solve the conditions

Effectiveness

- Test on known vulnerabilities
  - Correct the vulnerable points in PoC files
  - Test if we could find the same vulnerabilities
- Discover unknown vulnerabilities
  - Representative samples

Known vulnerabilities (8 vuls in 8 apps)

ID	Application	version	input
CVE-2010-1932	Xnview	1.97.4	mbm
CVE-2011-5233	irfanview	4.30	tif
OSVDB-83812	ZipItFast	3.0 pro	zip
CVE-2014-1761	Microsoft Word	2010	rtf
EDB-ID-39353	VLC	2.2.1	mp4
EDB-ID-17363	1ClickUnzip	3.0.0	zip
CVE-2010-2553	MediaPlayer	9.00.00.4503	avi
CVE-2015-0327	Adobe Flash	13sa	swf

Unknown vulnerabilities (47 vuls in 17 apps)

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

Evaluation

Details

- We have discovered 6 vulnerabilities of 8 known ones.
- CVE-2015-0327: overwrite the program function with scripts and change the behavior
- CVE-2010-2553: multiple level nested loops of accesses

ID	record-replay phase				analysis phase			resolve phase		
	snapshot size	changelog size	record time	replay time	trace size	trace #instr.	analy. time	relev. #instr.	constaint file size	extr. time
CVE-2010-1932	430.6MB	36.6MB	29s	486s	2.8GB	12.3M	99s	619	192KB	37s
CVE-2011-5233	516.1MB	18.5MB	37s	738s	9.8GB	43.9M	112s	795	251KB	96s
OSVDB-83812	819.3MB	13.6MB	83s	1257s	31.9GB	142.5M	787s	10	4.3kB	52s
CVE-2014-1761	855.3MB	52.3MB	178s	3712s	205.8GB	918.6M	6478s	183	17.8KB	198s
EDB-ID-39353	507.6MB	15.0MB	62s	271s	8.2GB	36.7M	10s	3082	331.1KB	674s
EDB-ID-17363	500.2MB	32.6MB	70s	889s	3.1GB	13.7M	45s	2313	191.2KB	502s
CVE-2010-2553	282.5MB	22.9MB	100s	806s	10.7GB	47.6M	565s	-	-	-
CVE-2015-0327	610.8MB	13.8MB	34s	682s	25.2GB	112.4M	709s	-	-	-

Case studies

- Tainted access offset
  - offset is influenced by input

```
dst address = Object + offset
memcpy(Dst, Src)
```

- Implicit taint situations
  - Allocation side: Size of malloc is based on the length of inputs
  - Access side: Number of data copy is based on keywords

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

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

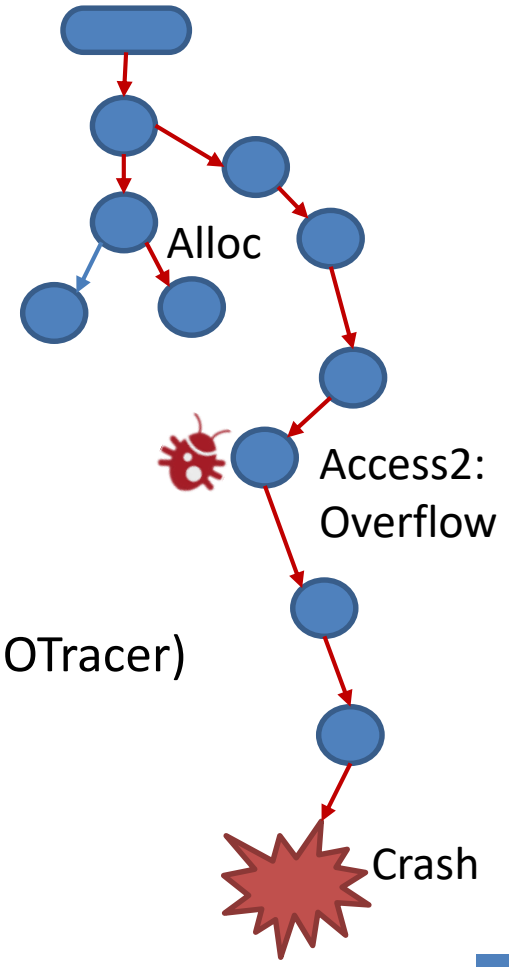
Case studies

- Multiple vulnerabilities in one trace
  - We found two vulnerable points when testing the PoC of CVE-2014-1761
- Long testing time
  - A VLC vulnerability happened at the end of playing for more than several minutes

Evaluation

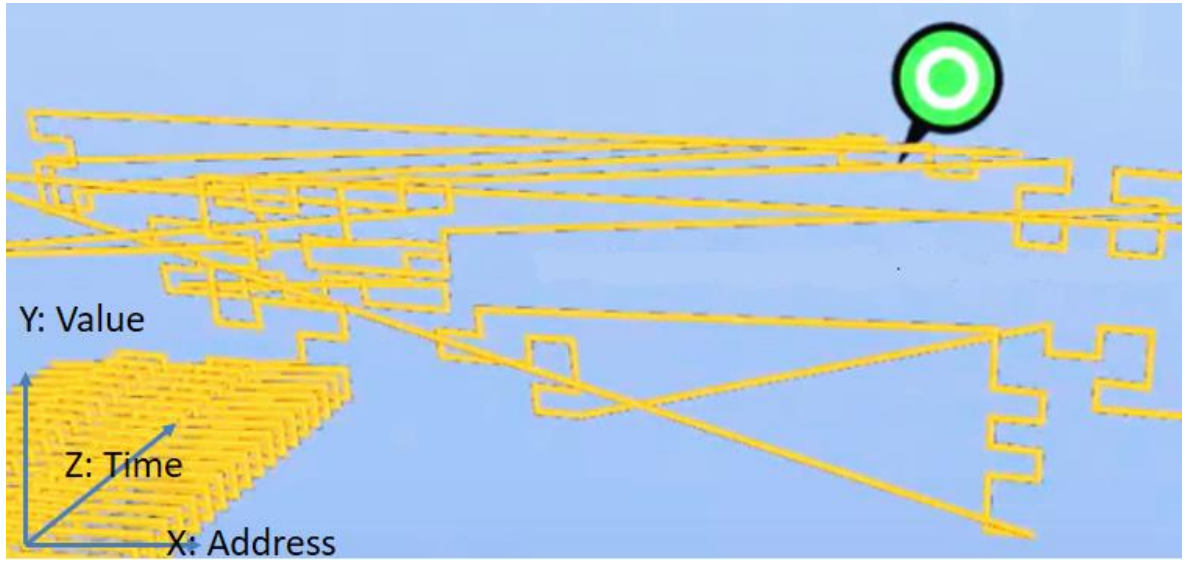
Comparison

- WinAFL
  - Need more knowledge of the target software and its functions
  - Fail to find new vulnerabilities
- Radamsa
  - Black-box fuzzer with various kinds of heuristics and change patterns
  - Potplayer: 1144 crashes, 11 crash points, 3 vulnerable points (included by HOTracer)



Future Work

- Custom heap management functions
- Complicated heap access operations
- Constraint simplification and solve
- Exploitability assess
- Use after free



Conclusion

- We proposed a new offline dynamic analysis solution to discover heap overflow vulnerabilities.
- We pointed out the root cause of heap overflow vulnerabilities is the inconsistency between heap operations.
- We addressed several challenges to make the solution practical and efficient.
- We implemented a prototype system, HOTracer, and found 47 previously unknown heap vulnerabilities in 17 applications with it.

Thank you

Xiangkun Jia
jiaxiangkun@tca.iscas.ac.cn