KOOLE: Towards Facilitating Exploit Generation of Kernel Out-Of-Bounds Write Vulnerabilities

Weiteng Chen, Xiaozhen Zou, Guoren Li, Zhiyun Qian
University of California, Riverside
Introduction

• According to syzbot, Google’s kernel fuzzing platform, there were 1216 Linux kernel bugs discovered and fixed during a single year (from Aug 2017 to Sep 2018).

Introduction

• According to syzbot, Google’s kernel fuzzing platform, there were 1216 Linux kernel bugs discovered and fixed during a single year (from Aug 2017 to Sep 2018).

• One promising direction is to automate the exploit generation and prioritize those exploitable.
Motivating Examples

1. struct Type1 { ...; }
2. struct Type2 { Type1 sk; uint64_t option; ...; }
3. struct Type3 { int (*ptr)(); ...; }
4. struct Type4 { uint64_t state; Type3 *sk; ...; }
5. struct Type5 { atomic_t refcnt; ...; }
6. Type2 gsock = { ... , .option = 0x08080000000000000000, };
7. Type1 * vul = NULL; Type3 * tgt = NULL;
8. void sys_socket() //sizeof(Type1) == sizeof(Type3)
9. vul = kmalloc(sizeof(Type1))

1. Allocate the vulnerable object

10. void sys_accept()
11. vul = (Type2*)vul; //type confusion
12. vul->option = gsock.option; //Vulnerability Point

13. void sys_setsockopt(val) //not invoked in given PoC
14. if (val == -1) return;
15. gsock.option = val;

CVE-2018-5703
Motivating Examples

1. struct Type1 { ...; }
2. struct Type2 { Type1 sk; uint64_t option; ...; }
3. struct Type3 { int (*ptr)(); ...; }
4. struct Type4 { uint64_t state; Type3 *sk; ...; }
5. struct Type5 { atomic_t refcnt; ...; }
6. Type2 gsock = { ..., .option = 0x0808000000000000, };  
7. Type1 * vul = NULL; Type3 * tgt = NULL;
8. void sys_socket() //sizeof(Type1) == sizeof(Type3)
9. vul = kmalloc(sizeof(Type1))
10. void sys_accept()
11. vul = (Type2*)vul; //type confusion
12. vul->option = gsock.option; //Vulnerability Point
13. void sys_setsockopt(val) //not invoked in given PoC
14. if (val == -1) return;
15. gsock.option = val;

CVE-2018-5703
Motivating Examples

1. struct Type1 { ...};
2. struct Type2 { Type1 sk; uint64_t option; ...};
3. struct Type3 { int (*ptr)(); ...};
4. struct Type4 { uint64_t state; Type3 *sk; ...};
5. struct Type5 { atomic_t refcnt; ...};
6. Type2 gsock = { ..., .option = 0x0808000000000000, };  
7. Type1 * vul = NULL; Type3 * tgt = NULL;
8. void sys_socket() //sizeof(Type1) == sizeof(Type3)
9. vul = kmalloc(sizeof(Type1))  
10. void sys_accept()
11. vul = (Type2*)vul;  //type confusion
12. vul->option = gsock.option;  //Vulnerability Point
13. void sys_setsockopt(val) //not invoked in given PoC
14. if (val == -1) return;
15. gsock.option = val;

1. Allocate the vulnerable object
2. Overwrite the following object via the vulnerable object
3. Control the overflown data

CVE-2018-5703
Motivating Examples

1. struct Type1 { ...};  
2. struct Type2 { Type1 sk; uint64_t option; ... };  
3. struct Type3 { int (*ptr)(); ... };  
4. struct Type4 { uint64_t state; Type3 *sk; ... };  
5. struct Type5 { atomic_t refcnt; ... };  
6. Type2 gsock = { ..., .option = 0x0808000000000000, };  
7. Type1 * vul = NULL; Type3 * tgt = NULL;  
8. void sys_socket() //sizeof(Type1) == sizeof(Type3)  
9.   vul = kmalloc(sizeof(Type1))  
10. void sys_accept()  
11.   vul = (Type2*)vul;  
12.   vul->option = gsock.option; //Vulnerability Point  
13. void sys_setsockopt(val) //not invoked in given PoC  
14.   if (val == -1) return;  
15.   gsock.option = val;  

CVE-2018-5703
Challenge 1: Exploration

• The initial PoC does not manifest the complete capability the corresponding vulnerability has.
Challenge 2: Modeling Capability

```c
void example1(size)
    vul = kmalloc(size);
    vul[size] = '\0';
CVE-2016-6187

void example2(i)
    vul = (char*)kmalloc(sizeof(TYPE));
    //omit other OOB points on the path
    vul[i/8] |= 1<<(i&0x7);//set 1 bit
CVE-2017-7184
```
Challenge: Modeling Capability

void example1(size)
    vul = kmalloc(size);
    vul[size] = '\0';
CVE-2016-6187

different OOB vulnerability instances exhibit a wide range of capabilities, in terms of:
1) how far the write can reach
2) how many bytes can be written
3) and what value can be written.

void example2(i)
    vul = (char*)kmalloc(sizeof(TYPE));
    //omit other OOB points on the path
    vul[i/8] |= 1<<(i&0x7);//set 1 bit
CVE-2017-7184
Challenge 3: Exploitability Evaluation

Vuln Object’s size: $s = \text{sizeof}(\text{Type1})$
Offset: $o = \text{sizeof}(\text{Type1})$
Length: $l = 8$ bytes
Value: $v = 0 \sim 0xffffffffffffffff$
Challenge 3: Exploitability Evaluation

Vulnerable Object’s size: $s = \text{sizeof}(\text{Type1})$
Offset: $o = \text{sizeof}(\text{Type1})$
Length: $l = 8$ bytes
Value: $v = 0 \sim 0xffffffffffffffff$

1. struct Type1 { ...; };
2. struct Type2 { Type1 sk; uint64_t option; ...; };
3. struct Type3 { int (*ptr)(); ...; };
4. struct Type4 { uint64_t state; Type3 *sk; ...; };
5. struct Type5 { atomic_t refcnt; ...; };
6. Type2 gsock = { ...; .option = 0x0808000000000000, };  
7. Type1 * vul = NULL; Type3 * tgt = NULL;
8. void sys_socket() //sizeof(Type1) == sizeof(Type3)
9. vul = kmalloc(sizeof(Type1))
... ...
16. void sys_create_tgt()
17. tgt = kmalloc(sizeof(Type3));
18. tgt->ptr = NULL; //init ptr
19. void sys_deref() { if (tgt->ptr) tgt->ptr(); }  ➔ IP hijacking

Our target object with a function pointer at the beginning

Vulnerable Object

Target Object

Overwritten value

Size

The length of this overflow

Offset

Vuln Object’s size: $s = \text{sizeof}(\text{Type1})$
Offset: $o = \text{sizeof}(\text{Type1})$
Length: $l = 8$ bytes
Value: $v = 0 \sim 0xffffffffffffffff$
Challenge 3: Exploitability Evaluation

Vuln Object’s size: \( s = \text{sizeof} \)\( \text{Type1} \)
Offset: \( o = \text{sizeof} \)\( \text{Type1} \)
Length: \( l = 8 \) bytes
Value: \( v = 0 \sim 0xffffffffffffffff \)

1. Heap layout arrangement: Heap Feng Shui
Challenge 3: Exploitability Evaluation

Vuln Object’s size: $s = \text{sizeof}(\text{Type1})$
Offset: $o = \text{sizeof}(\text{Type1})$
Length: $l = 8$ bytes
Value: $v = 0 \sim 0xffffffffffffffff$

1. Heap layout arrangement: Heap Feng Shui
2. How to evaluate exploitability against different target objects?
3. How to efficiently search for suitable target objects among hundreds of candidates?
Scope and Assumption

• Only generate exploit primitives to achieve IP hijacking
• Modern defenses are out of scope
  • Kernel Address Space Layout Randomization (KASLR)
  • Supervisor Mode Execution Prevention (SMEP)
  • Supervisor Mode Access Prevention (SMAP)
• Only encode some well-known heap Feng Shui strategies
Overview

- Vulnerability Analysis
- Capability Summarization
- Exploitability Evaluation
- Capability Exploration
Overview

• Vulnerability Analysis
• Capability Summarization
• Exploitability Evaluation
• Capability Exploration
Overview

- Vulnerability Analysis
- Capability Summarization
- Exploitability Evaluation
- Capability Exploration
Overview

- Vulnerability Analysis
- Capability Summarization
- Exploitability Evaluation
- Capability Exploration
Overview

- Vulnerability Analysis
- Capability Summarization
- Exploitability Evaluation
- Capability Exploration
Capability Summarization

• For each OOB write, we use a 3-tuple (offset, length, overwritten value) to present it.
  • offset, length and overwritten value are all symbolic expression.
  • 11. \texttt{vul} = (\texttt{Type2*})\texttt{vul}; \quad \text{//type confusion}
  12. \texttt{vul->option} = \texttt{gsock.option}; \quad \text{//Vulnerability Point}

```
offset = sizeof(Type1)  length = 8 bytes
Overwritten value = gsock.option
```

Vulnerable Object

Target Object
Capability Summarization

• For each OOB write, we use a 3-tuple (offset, length, overwritten value) to present it.
  • offset, length and overwritten value are all symbolic expression.
  • 11. vul = (Type2*)vul;     //type confusion
      12. vul->option = gsock.option; //Vulnerability Point

• For a single path, we consider a set of OOB write summarization.
Capability Summarization

- **Path constraints** and the **size of the vulnerable object** that are coupled with OOB writes should be included.

13. void sys_setsockopt(val) //not invoked in given PoC
14. if (val == -1) return;
15. gsock.option = val;
Capability Exploration: Capability-Guided Fuzzing

- Existing coverage-guided fuzzers are insensitive to the capability.
- We use dynamic instrumentation to hook all the vulnerability points to collect information (e.g., offset, length, value) as the feedback.
Exploitability Evaluation: Target Objects

1. Function Pointer
2. Data Pointer
3. Non-Pointer:
   1. Uid
   2. Reference Counter

```c
struct Type3 { int (*ptr)(); ...; }

16. void sys_create_tgt()
17. tgt = kmalloc(sizeof(Type3));
18. tgt->ptr = NULL;
19. void sys_deref() { if (tgt->ptr) tgt->ptr(); }
```

```json
{
  "Type3": {
    "type": "function pointer",
    "offset": 0, // the offset to the function pointer
    "size": 192, // size of this object
    "payload": 0xdeadbeef,
    "allocate": // How to allocate this object
      "sys_create_tgt()",
    "deref": // How to trigger the dereference of the function pointer
      "sys_deref();"
  }
}
```
Exploitability Evaluation

1. Construct a memory model
2. Update the memory model with OOB writes
3. Query SMT solver with respect to path constraints

```
struct Type3 {
  int (*ptr)();
  ...
};
```
Exploitability Evaluation

1. Construct a memory model
2. Update the memory model with OOB writes
3. Query SMT solver with respect to path constraints

Memory Model

- **Vulnerable Object**
- **Target Object**

offset = sizeof(Type1)

length = 8 bytes

object size = sizeof(Type1)

type4.ptr == diverted address??

struct Type4 {
  uint64_t state;
  int (*ptr)();
  ...;
};
Evaluation

• Dataset and Setup
  • 7 from CVE database
  • 10 from syzbot (a fuzzing platform based on Syzkaller)
Evaluation

• Dataset and Setup
  • 7 from CVE database
  • 10 from syzbot (a fuzzing platform based on Syzkaller)

• Results

<table>
<thead>
<tr>
<th>CVE-ID</th>
<th>Race Condition</th>
<th>#public EXP</th>
<th>#generated EXP*</th>
</tr>
</thead>
<tbody>
<tr>
<td>CVE-2016-6187</td>
<td>No</td>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>CVE-2016-6516</td>
<td>Yes</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>CVE-2017-7184</td>
<td>No</td>
<td>1</td>
<td>3</td>
</tr>
<tr>
<td>CVE-2017-7308</td>
<td>No</td>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>CVE-2017-7533</td>
<td>Yes</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>CVE-2017-1000112</td>
<td>No</td>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>CVE-2018-5703</td>
<td>No</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>Overall</td>
<td></td>
<td>4</td>
<td>11</td>
</tr>
</tbody>
</table>

*: We count the number of distinct exploits based on the target object we exploit.
## Evaluation

<table>
<thead>
<tr>
<th>Commit Hash</th>
<th>#public EXP</th>
<th>#generated EXP</th>
</tr>
</thead>
<tbody>
<tr>
<td>813961de3ee6474dd5703e883471fd941d6c8f69</td>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>35f7d5225ffcbf1b759f641aec1735e3a89b1914</td>
<td>0</td>
<td>2</td>
</tr>
<tr>
<td>bbeb6e4323dad9b5e0ee9f60c223dd532e2403b1</td>
<td>0</td>
<td>2</td>
</tr>
<tr>
<td>eb73190f4fbeedf762394e92d6a4ec9ace684c88</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>4576cd469d980317c4edd9173f8b694aa71ea3a3</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>17cfe79a65f98abe535261856c5aef14f306dff7</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>9fa68f620041be04720d0cbfb1bd3ddf6310b24</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>3619dec5103dd999a777e3e4ea08c8f40a6ddc57</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>70303420b5721c38998cf987e6b7d30cc62d4ff1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>bb29648102335586e9a66289a1d98a0cb392b6e5</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Overall</td>
<td>1</td>
<td>8</td>
</tr>
<tr>
<td>CVE or Commit</td>
<td>Tracing</td>
<td>Solving</td>
</tr>
<tr>
<td>--------------</td>
<td>---------</td>
<td>---------</td>
</tr>
<tr>
<td>CVE-2016-6187</td>
<td>38s</td>
<td>1s</td>
</tr>
<tr>
<td>CVE-2017-7184</td>
<td>27s</td>
<td>45s</td>
</tr>
<tr>
<td>CVE-2017-7308</td>
<td>48s</td>
<td>4s</td>
</tr>
<tr>
<td>CVE-2017-7533</td>
<td>160s</td>
<td>164s</td>
</tr>
<tr>
<td>CVE-2017-1000112</td>
<td>36s</td>
<td>132s</td>
</tr>
<tr>
<td>CVE-2018-5703</td>
<td>85s</td>
<td>41s</td>
</tr>
<tr>
<td>813961de3ee6474dd570</td>
<td>34s</td>
<td>5s</td>
</tr>
<tr>
<td>35f7d5225ffcbf1b759f</td>
<td>34s</td>
<td>18s</td>
</tr>
<tr>
<td>bbeb6e4323dad9b5e0ee</td>
<td>48s</td>
<td>26s</td>
</tr>
<tr>
<td>eb73190f4fbeedf76239</td>
<td>54s</td>
<td>104s</td>
</tr>
<tr>
<td>4576cd469d980317c4ed</td>
<td>57s</td>
<td>7s</td>
</tr>
</tbody>
</table>

Note: We only apply fuzzing when necessary, i.e., our system is unable to find a suitable target object given the capability in the original PoC.
Discussion

• The principle of separating capability summarization from exploitability evaluation can be applied to other types of kernel vulnerabilities due to:
  • Large search space: the inherently multi-interaction nature of kernel.
  • Vulnerabilities sometimes can be converted from one type to another, but all of them require to corrupt the kernel data.
THANK YOU
ANY QUESTION?

Contact: wchen130@ucr.edu
Source Code: https://github.com/seclab-ucr/KOOBE