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

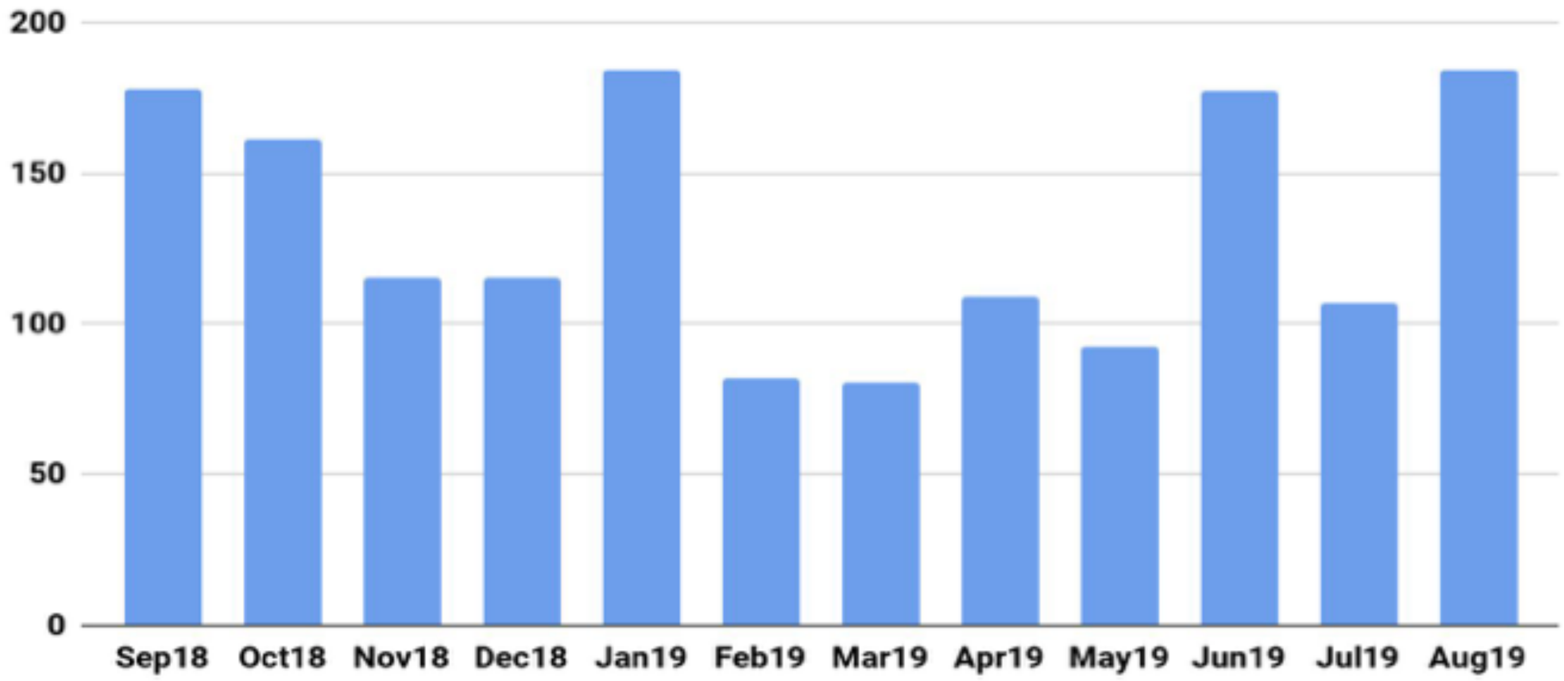
Weiteng Chen, Xiaozhen Zou, Guoren Li, Zhiyun Qian

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).

Bugs/Month [1]



[1] syzbot: update and open problems at Linux Plumbers 2019.
https://linuxplumbersconf.org/event/4/contributions/428/attachments/308/515/syzbot__Plumbers_2019.pdf

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

```
struct Type1 { ...;                                 };
struct Type2 { Type1 sk;  uint64_t option; ...;     };
struct Type3 { int (*ptr)(); ...;                   };
struct Type4 { uint64_t state; Type3 *sk; ...;      };
struct Type5 { atomic_t refcnt; ...;                };
Type2 gsock = { ..., .option = 0x0808000000000000,  };
Type1 * vul = NULL; Type3 * tgt = NULL;
void sys_socket()  //sizeof(Type1) == sizeof(Type3)
  vul = kmalloc(sizeof(Type1))

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

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

1. Allocate the vulnerable object

CVE-2018-5703

Motivating Examples

```
struct Type1 { …;                                    };
struct Type2 { Type1 sk;  uint64_t option; …;         };
struct Type3 { int (*ptr)(); …;                      };
struct Type4 { uint64_t state; Type3 *sk; …;         };
struct Type5 { atomic_t refcnt; …;                   };
Type2 gsock = { …, .option = 0x0808000000000000,     };
Type1 * vul = NULL; Type3 * tgt = NULL;
void sys_socket()  //sizeof(Type1) == sizeof(Type3)
  vul = kmalloc(sizeof(Type1))

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

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

1. Allocate the vulnerable object

2. Overwrite the following object via the vulnerable object

CVE-2018-5703

Motivating Examples

```
struct Type1 { …;                                  };
struct Type2 { Type1 sk;  uint64_t option; …;       };
struct Type3 { int (*ptr)(); …;                     };
struct Type4 { uint64_t state; Type3 *sk; …;        };
struct Type5 { atomic_t refcnt; …;                  };
Type2 gsock = { …, .option = 0x0808000000000000,    };
Type1 * vul = NULL; Type3 * tgt = NULL;
void sys_socket()  //sizeof(Type1) == sizeof(Type3)
  vul = kmalloc(sizeof(Type1))

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

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

1. Allocate the vulnerable object

2. Overwrite the following object via the vulnerable object

Control the overflown data

CVE-2018-5703

Motivating Examples

```
struct Type1 { …;                                    };
struct Type2 { Type1 sk;  uint64_t option; …;        };
struct Type3 { int (*ptr)(); …;                      };
struct Type4 { uint64_t state; Type3 *sk; …;         };
struct Type5 { atomic_t refcnt; …;                   };
Type2 gsock = { …, .option = 0x0808000000000000,   };
Type1 * vul = NULL; Type3 * tgt = NULL;
void sys_socket()  //sizeof(Type1) == sizeof(Type3)
  vul = kmalloc(sizeof(Type1))

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

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

1. Allocate the vulnerable object

2. Overwrite the following object via the vulnerable object

Control the overflown data

CVE-2018-5703

Challenge 1: Exploration

- The initial PoC does not manifest the complete capability the corresponding vulnerability has.

Challenge 2: Modeling Capability

```
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

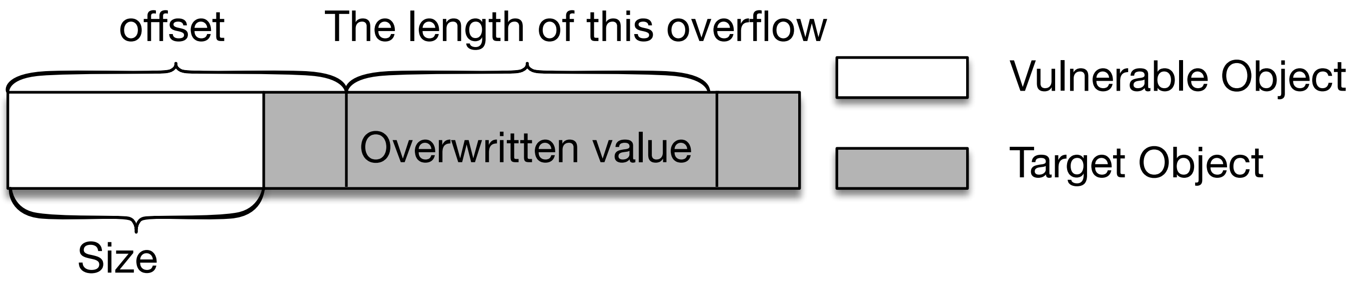
```
void example2(i)
  vul = (char*)kmalloc(sizeof(TYPE));
  //omit other OOB points on the path
  vul[i/8] |= 1<<(i&0x7);//set 1 bit
```

CVE-2017-7184

different OOB vulnerability instances exhibit a wide range of capabilities, in terms of:

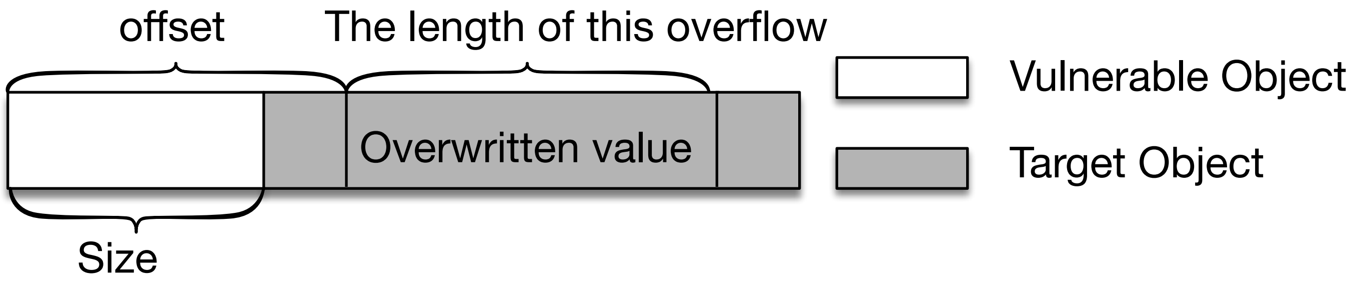
1) **how far the write can reach**
2) **how many bytes can be written**
3) **and what value can be written.**

Challenge 3: Exploitability Evaluation



Vuln Object's size: s = sizeof(Type1)
Offset: o = sizeof(Type1)
Length: l = 8 bytes
Value: v = 0 ~ 0xffffffffffffffff

Challenge 3: Exploitability Evaluation



Vuln Object's size: s = sizeof(Type1)
Offset: o = sizeof(Type1)
Length: l = 8 bytes
Value: v = 0 ~ 0xffffffffffffffff

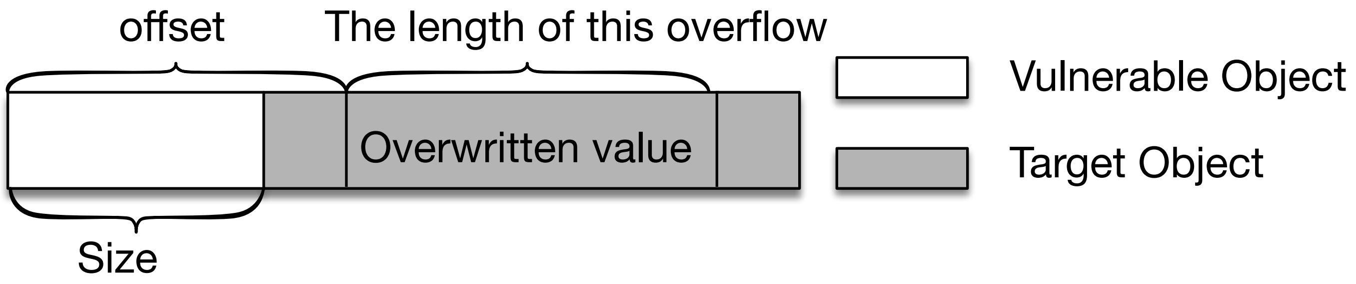
```
struct Type1 { …;                                    };
struct Type2 { Type1 sk;  uint64_t option; …;        };
struct Type3 { int (*ptr)(); …;                      };
struct Type4 { uint64_t state; Type3 *sk; …;         };
struct Type5 { atomic_t refcnt; …;                   };
Type2 gsock = { …, .option = 0x0808000000000000,     };
Type1 * vul = NULL; Type3 * tgt = NULL;
void sys_socket()  //sizeof(Type1) == sizeof(Type3)
  vul = kmalloc(sizeof(Type1))
… …
void sys_create_tgt()
  tgt = kmalloc(sizeof(Type3));
  tgt->ptr = NULL;                  //init ptr

void sys_deref() { if (tgt->ptr) tgt->ptr(); }
```

Our target object with a function pointer at the beginning

IP hijacking

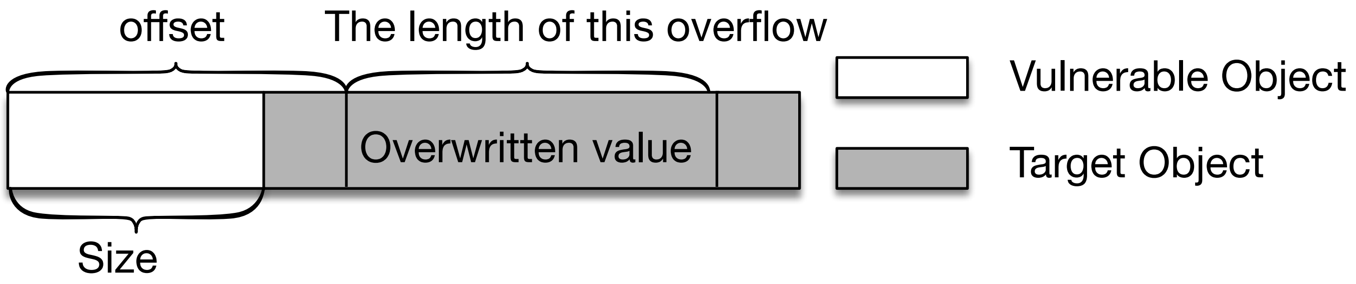
Challenge 3: Exploitability Evaluation



Vuln Object's size: s = sizeof(Type1)
Offset: o = sizeof(Type1)
Length: l = 8 bytes
Value: v = 0 ~ 0xffffffffffffffff

1. Heap layout arrangement: Heap Feng Shui

Challenge 3: Exploitability Evaluation



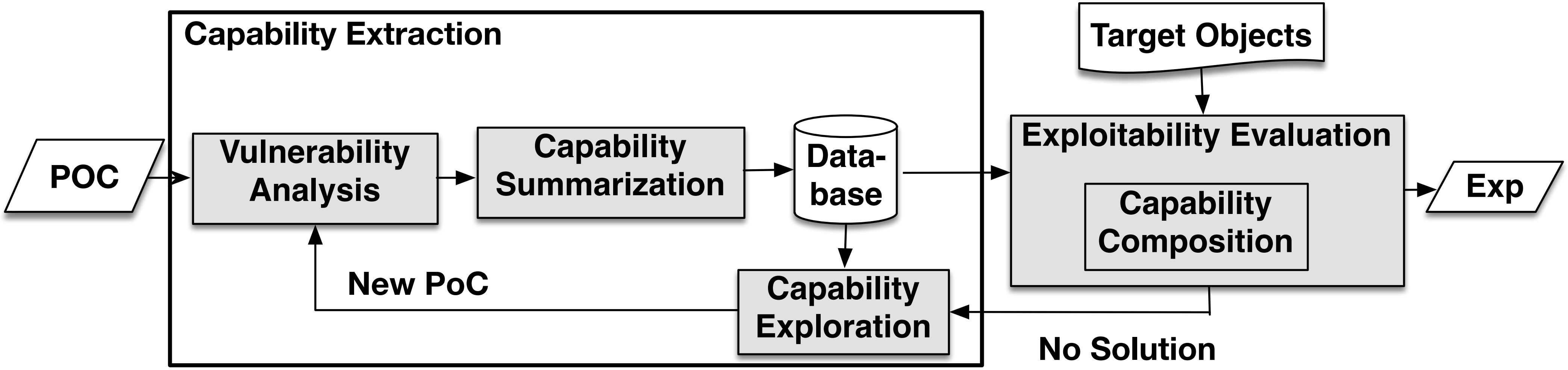
Vuln Object's size: s = sizeof(Type1)
Offset: o = sizeof(Type1)
Length: l = 8 bytes
Value: v = 0 ~ 0xffffffffffffffff

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

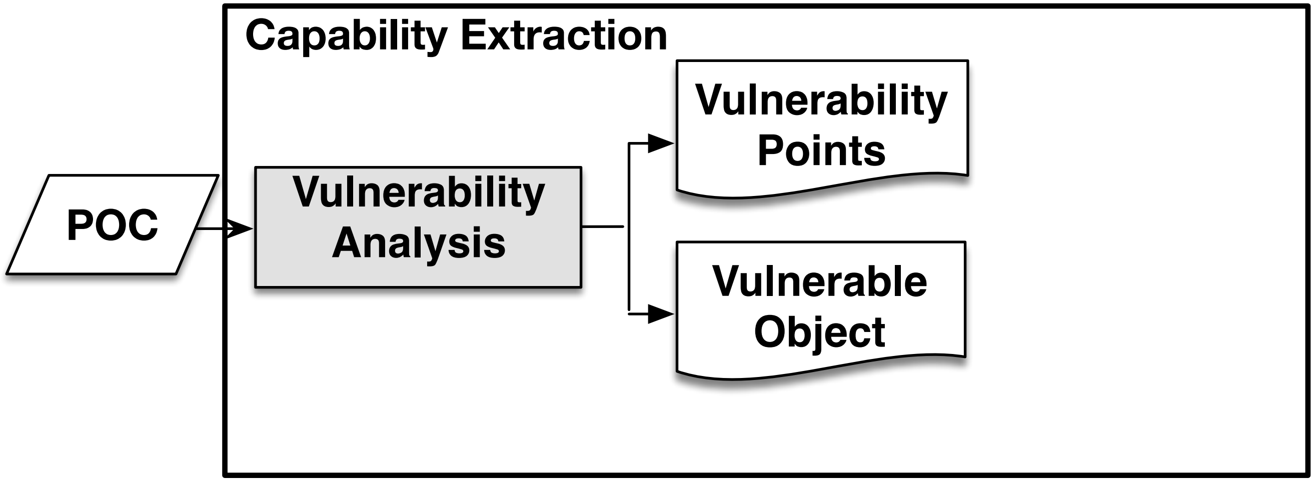
- Only generate exploit primitives to achieve IP hijacking
- Modern defenses are out of scope
  - Kernel Address Space Layout Randomization (KASLR)
  - Supervisor Mode Execution Prevention (SMEP)
  - Supervisor Mode Access Prevention (SMAP)
- Only encode some well-known heap Feng Shui strategies

Overview



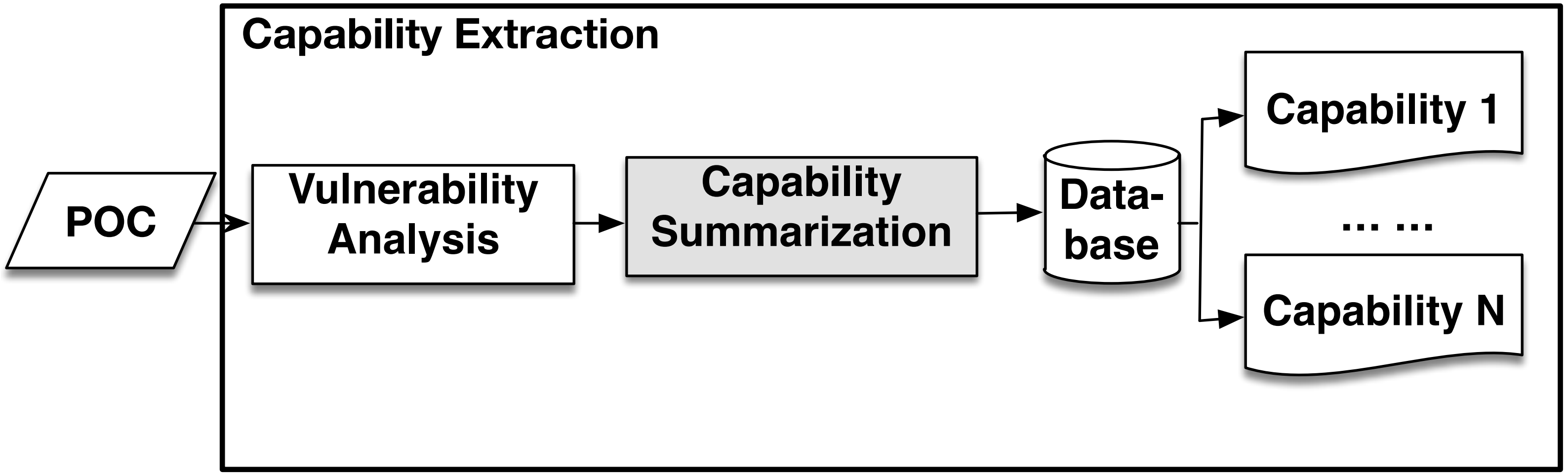
- Vulnerability Analysis
- Capability Summarization
- Exploitability Evaluation
- Capability Exploration

Overview



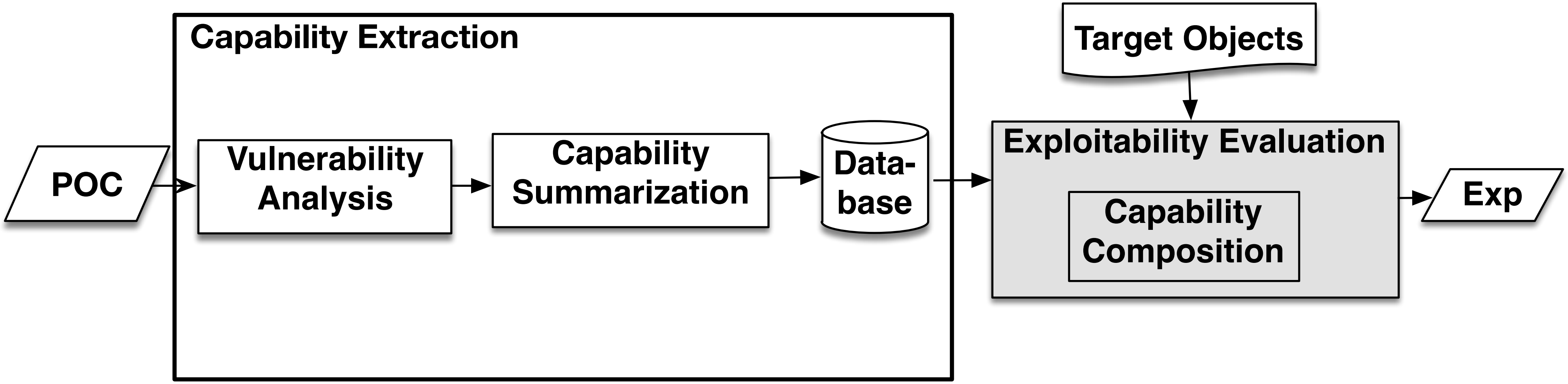
- Vulnerability Analysis
- Capability Summarization
- Exploitability Evaluation
- Capability Exploration

Overview



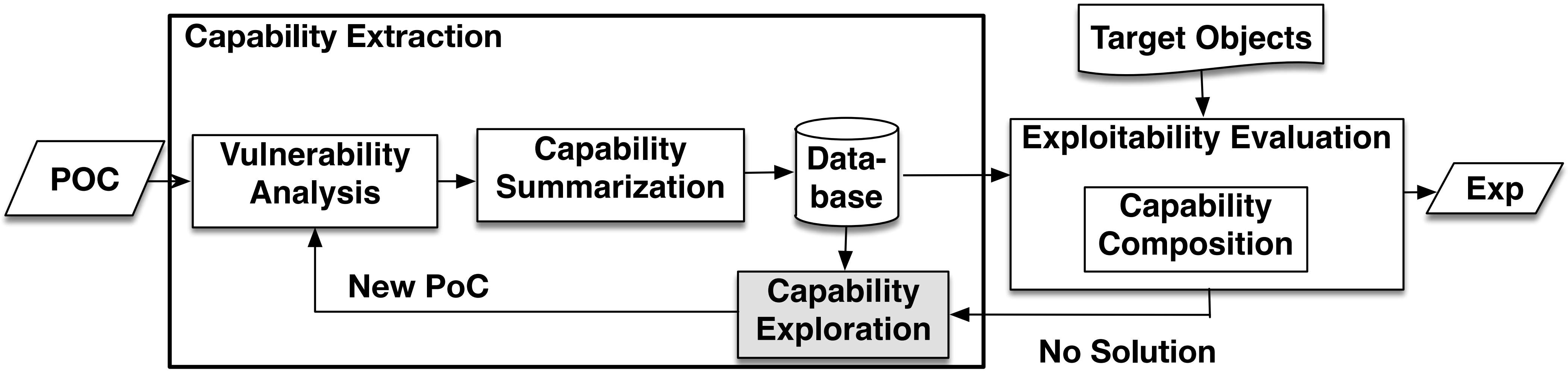
- Vulnerability Analysis
- Capability Summarization
- Exploitability Evaluation
- Capability Exploration

Overview



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

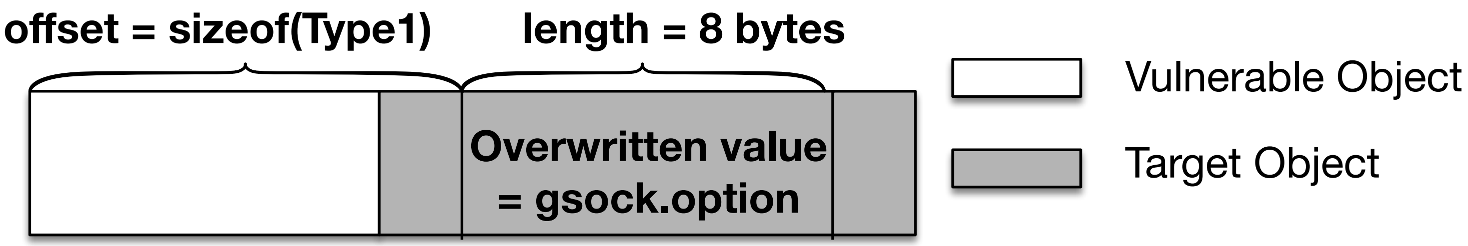
Overview



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

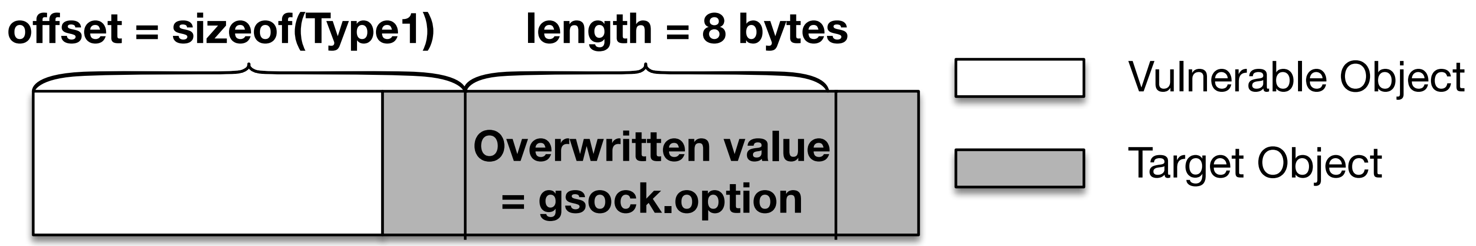
Capability Summarization

- For each OOB write, we use a 3-tuple (***offet, 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
    ```


    ```

Capability Summarization

- For each OOB write, we use a 3-tuple (***offet***, ***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.
    ```

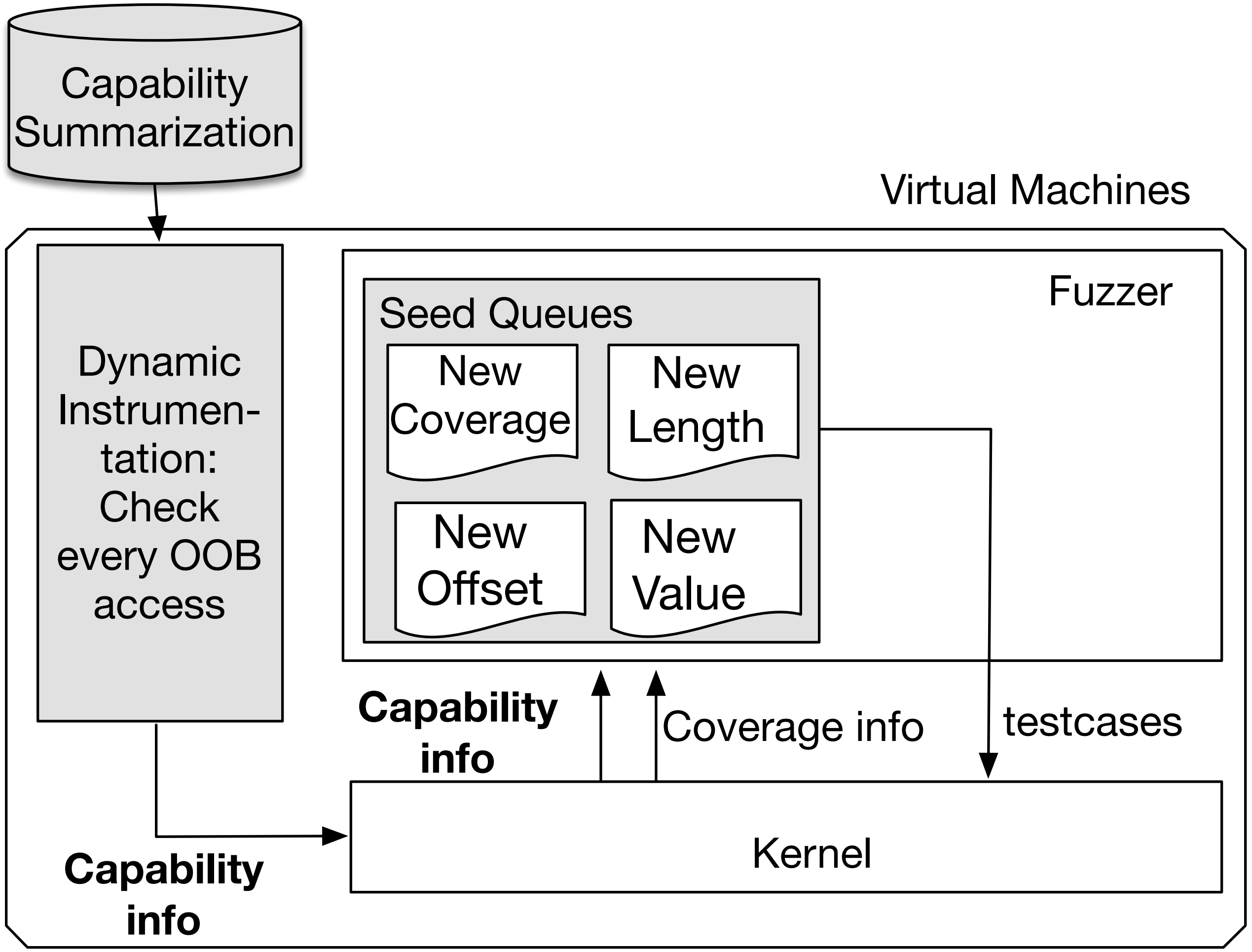
Capability Summarization

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

```
void sys_setsockopt(val) //not invoked in given PoC
  if (val == -1) return;
  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.



Fuzzer's Architecture

Exploitability Evaluation: Target Objects

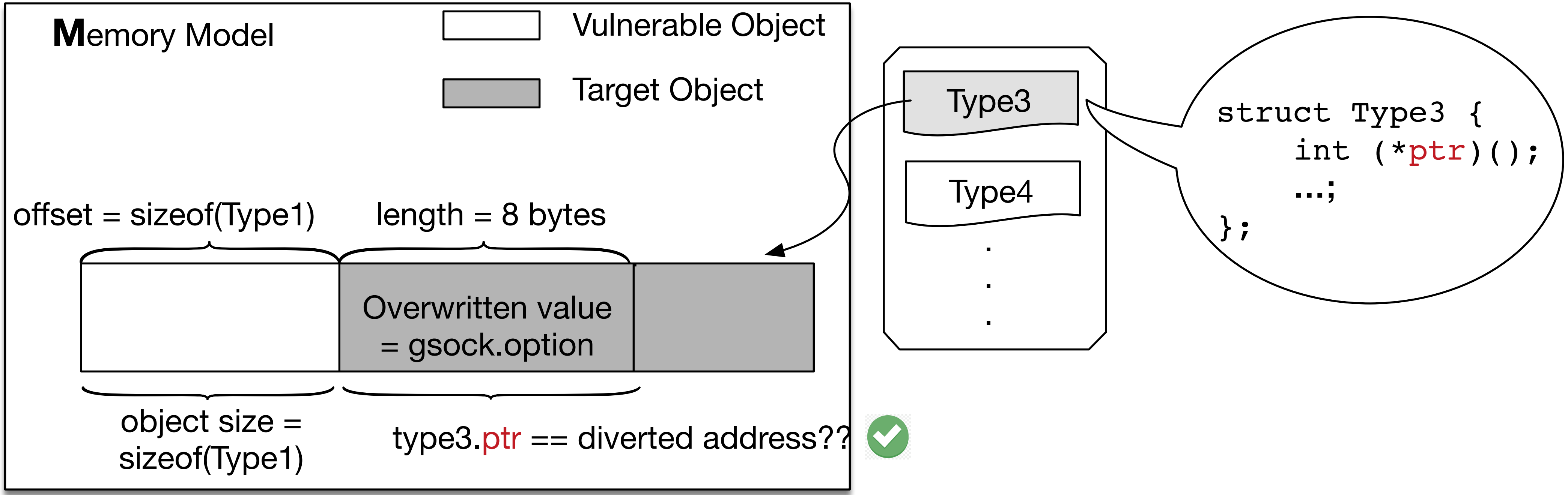
1. **Function Pointer**
2. **Data Pointer**
3. **Non-Pointer:**
   1. **Uid**
   2. **Reference Counter**

```
struct Type3 { int (*ptr)(); …;                        };

void sys_create_tgt()
  tgt = kmalloc(sizeof(Type3));
  tgt->ptr = NULL;
void sys_deref() { if (tgt->ptr) tgt->ptr(); }
```

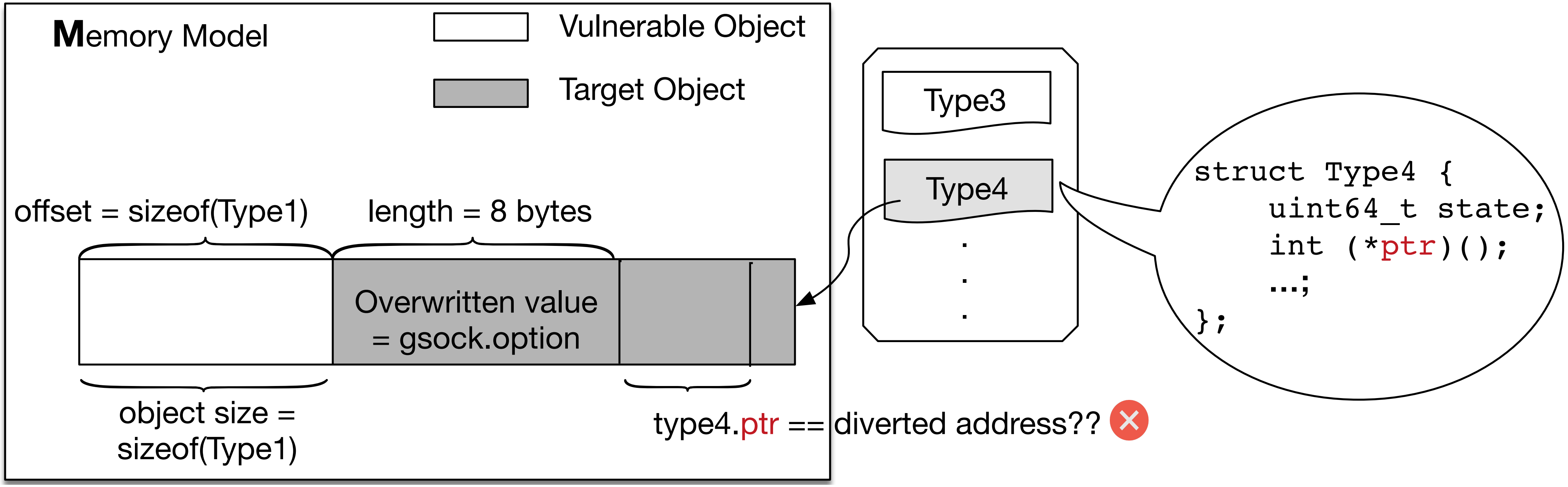
```
{
   "Type3": {
      "type":  "function pointer",
      "offset":  0,  // the offset to the function pointer
      "size":  192,  // size of this object
      "payload":  0xdeadbeef,
      "allocate":  // How to allocate this object
         "sys_create_tgt()",
      "deref": // How to trigger the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

Exploitability Evaluation



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

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

CVE-ID	Race Condition	#public EXP	#generated EXP*
CVE-2016-6187	No	1	2
CVE-2016-6516	Yes	0	0
CVE-2017-7184	No	1	3
CVE-2017-7308	No	1	2
CVE-2017-7533	Yes	0	1
CVE-2017-1000112	No	1	2
CVE-2018-5703	No	0	1
Overall		4	11

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

Evaluation

Commit Hash	#public EXP	#generated EXP
813961de3ee6474dd5703e883471fd941d6c8f69	1	2
35f7d5225ffcbf1b759f641aec1735e3a89b1914	0	2
bbeb6e4323dad9b5e0ee9f60c223dd532e2403b1	0	2
eb73190f4fbeedf762394e92d6a4ec9ace684c88	0	1
4576cd469d980317c4edd9173f8b694aa71ea3a3	0	1
17cfe79a65f98abe535261856c5aef14f306dff7	0	0
9fa68f620041be04720d0cbfb1bd3ddfc6310b24	0	0
3619dec5103dd999a777e3e4ea08c8f40a6ddc57	0	0
70303420b5721c38998cf987e6b7d30cc62d4ff1	0	0
bb29648102335586e9a66289a1d98a0cb392b6e5	0	0
Overall	1	8

Time Cost

CVE or Commit	Tracing	Solving	Fuzzing	Time to Patch
CVE-2016-6187	38s	1s	NA	NA
CVE-2017-7184	27s	45s	23m	14 days (estimated)
CVE-2017-7308	48s	4s	NA	NA
CVE-2017-7533	160s	164s	NA	NA
CVE-2017-1000112	36s	132s	NA	NA
CVE-2018-5703	85s	41s	194m	114 days
813961de3ee6474dd570	34s	5s	NA	12 days
35f7d5225ffcbf1b759f	34s	18s	8m	50 days
bbeb6e4323dad9b5e0ee	48s	26s	23m	11 days
eb73190f4fbeedf76239	54s	104s	NA	38 days
4576cd469d980317c4ed	57s	7s	NA	31 days

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