KLEAK

Practical Kernel Memory Disclosure Detection

Thomas Barabosch
Fraunhofer FKIE
thomas.barabosch@fkie.fraunhofer.de

Maxime Villard
NetBSD
m00nbsd.net
System Call

- Recap

Program

- Program continues normal execution
- calls syscall X

Syscall Dispatcher

- Syscall returns to user
- additional preparation

Syscall X

User Space
Kernel Space
Kernel/User Space Data Exchange
- copyin

- User space programs can not directly write to kernel space
- It points the kernel to a buffer, the kernel fetches this data
  - copyin, copyinstr, …
Kernel/User Space Data Exchange
- copyout

- Kernel uses dedicated functions to copy data to user space
  - copyout, copyoutstr, ...

- Supervisor Mode Access Prevention (SMAP)
Kernel Memory Disclosure (KMD)
- What is it?

- Inadvertently writing data from kernel to user space
- As a consequence a KMD may leak
  - random data
  - kernel pointers
  - keys/tokens /…
- KMDs typically do not lead to privilege escalation
  - However: they are an important step towards this goal!
int sys_getcontext(struct thread *td, 
    struct getcontext_args *uap) { 
    ucontext_t uc; 
    int ret; 

    if (uap->ucp == NULL) 
        ret = EINVAL; 
    else { 
        get_mcontext(td, 
                     &uc.uc_mcontext, 
                     GET_MC_CLEAR_RET); 
        PROC_LOCK(td->td_proc); 
        uc.uc_sigmask = td->td_sigmask; 
        PROC_UNLOCK(td->td_proc); 
        ret = copyout(&uc, uap->ucp, UC_COPY_SIZE); 
    } 
    return (ret); 
}
`sys_getcontext` function:

```c
int sys_getcontext(struct thread *td, 
                   struct getcontext_args *uap)
{
    ucontext_t uc;
    int ret;

    if (uap->ucp == NULL)
        ret = EINVAL;
    else {
        get_mcontext(td,
                     &uc.uc_mcontext,
                     GET_MC_CLEAR_RET);
        PROC_LOCK(td->td_proc);
        uc.uc_sigmask = td->td_sigmask;
        PROC_UNLOCK(td->td_proc);
        ret = copyout(&uc, uap->ucp, UC_COPY_SIZE);
    }
    return (ret);
}
```
Kernel Memory Disclosure
- CVE-2018-17155 / FreeBSD-EN-18:12.mem

```c
int sys_getcontext(struct thread *td,
                   struct getcontext_args *uap) {
    ucontext_t uc;
    int ret;

    if (uap->ucp == NULL)
        ret = EINVAL;
    else {
        get_mcontext(td,
                     &uc.uc_mcontext,
                     GET_MC_CLEAR_RET);
        PROC_LOCK(td->td_proc);
        uc.uc_sigmask = td->td_sigmask;
        PROC_UNLOCK(td->td_proc);
        ret = copyout(&uc, uap->ucp, UC_COPY_SIZE);
    }
    return (ret);
}
```
int sys_getcontext(struct thread *td,
        struct getcontext_args *uap) {
    ucontext_t uc;
    int ret;

    if (uap->ucp == NULL)
        ret = EINVAL;
    else {
        get_mcontext(td,
            &uc.uc_mcontext,
            GET_MC_CLEAR_RET);
        PROC_LOCK(td->td_proc);
        uc.uc_sigmask = td->td_sigmask;
        PROC_UNLOCK(td->td_proc);
        ret = copyout(&uc, uap->ucp, UC_COPY_SIZE);
    }
    return (ret);
}
int sys_getcontext(struct thread *td,  
  struct getcontext_args *uap) {
  ucontext_t uc;
  int ret;
  
  if (uap->ucp == NULL)
    ret = EINVAL;
  else {
    get_mcontext(td,  
      &uc.uc_mcontext,  
      GET_MC_CLEAR_RET);
    PROC_LOCK(td->td_proc);
    uc.uc_sigmask = td->td_sigmask;
    PROC_UNLOCK(td->td_proc);
    ret = copyout(&uc, uap->ucp, UC_COPY_SIZE);
  }
  return (ret);
}
struct ucontext4 {
    sigset_t uc_sigmask;
    struct mcontext4 uc_mcontext;
    struct ucontext4 *uc_link;
    stack_t uc_stack;
    int __spare__[8];
};
int sys_getcontext(struct thread *td,
    struct getcontext_args *uap) {
    ucontext_t uc;
    int ret;

    if (uap->ucp == NULL)
        ret = EINVAL;
    else {
        get_mcontext(td,
            &uc.uc_mcontext,
            GET_MC_CLEAR_RET);
        PROC_LOCK(td->td_proc);
        uc.uc_sigmask = td->td_sigmask;
        PROC_UNLOCK(td->td_proc);
        ret = copyout(&uc, uap->ucp, UC_COPY_SIZE);
    }
    return (ret);
}
Kernel Memory Disclosure
- CVE-2018-17155 / FreeBSD-EN-18:12.mem

```c
int sys_getcontext(struct thread *td,
                   struct getcontext_args *uap) {
    ucontext_t uc;
    int ret;

    if (uap->ucp == NULL)
        ret = EINVAL;
    else {
        get_mcontext(td,
                     &uc.uc_mcontext,
                     GET_MC_CLEAR_RET);
        PROC_LOCK(td->td_proc);
        uc.uc_sigmask = td->td_sigmask;
        PROC_UNLOCK(td->td_proc);
        ret = copyout(&uc, uap->ucp, UC_COPY_SIZE);
    }
    return (ret);
}
```

Leaked stack memory!
Kernel Memory Disclosure
- CVE-2018-17155 / FreeBSD-EN-18:12.mem

```c
int sys_getcontext(struct thread *td,
                   struct getcontext_args *uap) {
    ucontext_t uc;
    int ret;

    if (uap->ucp == NULL)
        ret = EINVAL;
    else {
        get_mcontext(td,
                     &uc.uc_mcontext,
                     GET_MC_CLEAR_RET);
        PROC_LOCK(td->td_proc);
        uc.uc_sigmask = td->td_sigmask;
        PROC_UNLOCK(td->td_proc);
        ret = copyout(&uc, uap->ucp, UC_COPY_SIZE);
    }
    return (ret);
}
```
int sys_getcontext(struct thread *td,
        struct getcontext_args *uap) {
    ucontext_t uc;
    int ret;
    
    if (uap->ucp == NULL)
        ret = EINVAL;
    else {
        get_mcontext(td,
              &uc.uc_mcontext,
              GET_MC_CLEAR_RET);
        PROC_LOCK(td->td_proc);
        uc.uc_sigmask = td->td_sigmask;
        PROC_UNLOCK(td->td_proc);
        ret = copyout(&uc, uap->ucp, UC_COPY_SIZE);
    }
    return (ret);
}

Leaks 2 ½ kernel pointers!

Leaked stack memory!
# Kernel Memory Disclosure

- CVE-2018-17155 / FreeBSD-EN-18:12.mem (fixed)

```c
int
sys_getcontext(struct thread *td, struct getcontext_args *uap)
{
    ucontext_t uc;
    int ret;

    if (uap->ucp == NULL)
        ret = EINVAL;
    else {
        bzero(&uc, sizeof(ucontext_t));
        get_mcontext(td, &uc.uc_mcontext, GET_MC_CLEAR_RET);
        PROC_LOCK(td->td_proc);
        uc.uc_sigmask = td->td_sigmask;
        PROC_UNLOCK(td->td_proc);
        ret = copyout(&uc, uap->ucp, UC_COPY_SIZE);
    }
    return (ret);
}
```
Kernel Memory Disclosure
- Why are they hard to detect?

- They are **silent** bugs
  - do not yield crashes
  - may be hidden by C libraries
- The **root of all evil**: the C programming language
- Current state of compilers
- System memory allocators (stack + heap)
- Architecture-dependent (e.g. i386 vs AMD64)
- Developers may be unaware of this issue
Kernel Memory Disclosure
- Typical error sources

- Refer to Mateusz Jurczyk’s publication about BochsPwn Reloaded!
- C language
  - Uninitialized variables
  - Struct alignment (arch-dependent)
  - Padding bytes in structs
  - Unions
- OS
  - Memory reuse in heap allocator/stack
  - Arbitrary syscall output buffers
Remember that
- local variables (stack) are uninitialized
- your heap implementation may return uninitialized data without flags like M_ZERO on FreeBSD

Do not trust your compiler!

Do not assume a certain architecture/padding/…!

Be really careful when dealing with structs/unions!

Initialize your data structures as early as possible!
Kernel Memory Disclosure
- How to avoid it? 2/2

- When developing a new syscall then dump the exchanged buffer in user space and check for leaked bytes.
- **Finally**: when in doubt zero out!
  - Security over efficiency
  - Remember: one leaked byte may break your KASLR!
Kernel Memory Disclosure  
- What to expect?

- Several researchers uncovered hundreds of KMDs
  - Lu et al. state that 37 KMDs were found in Linux in 2010/2011
  - Lu et al. found 19 KMDs in Linux and Android in 2016
  - Mateusz Jurczyk found 80 KMDs in Windows and Linux in 2018
  - ...

- So far no systematic investigation on the *BSDs
  - Exception: OpenBSD’s manual code reviews

- Assumption: there must be many KMDs in the *BSDs
  - Found some low-hanging fruits during manual code review in FreeBSD/NetBSD
  - Let’s patch the kernel to find more …
KLEAK
- Overview
KLEAK
- Overview

Diagram:

- Program
- Syscall Dispatcher
- User Space
- Kernel
- Stack
- Heap

Taints from program to syscall dispatcher to kernel.
KLEAK - Overview

Program

syscall dispatcher

syscall

memory allocations

stack

heap

User Space

Kernel
KLEAK
- Overview

![Diagram showing program, copyout, syscall dispatcher, syscall, stack, and heap.]

User Space

Kernel

stack

heap
KLEAK - Overview

User Space

Kernel

- stack
- heap

Program

- copyout
- syscall dispatcher
- syscall

copy out to user space
KLEAK - Overview

User Space

Kernel

- User Space
- Kernel

Program

Copyout

Kleak

Syscall dispatcher

Syscall

Stack

Heap

Copys out to user space
KLEAK
- Overview

User Space

Kernel

heap

stack

copyout

kleak

copys out to user space

program

syscall

dispatcher

hitmap

syscall
KLEAK
- Overview

User Space

Kernel

program

syscall dispatcher

syscall

returns to user space

stack

heap
KLEAN
- Tainting Memory Sources (Heap)

- We instrument the dynamic memory allocator to return marked chunks
  - `memset` the chunk with the marker byte
  - **Exception**: requested zero’d chunks

- In NetBSD, there are several ways to allocate dynamic memory
  - `malloc(9)`, `kmem_alloc`, pools (`uvm_km_alloc`)
  - First we did this in `malloc(9)`
  - Later we chose to taint the memory pools directly
KLEAK
- Tainting Memory Sources (Stack)

- Right before entering the syscall we taint the stack by allocating an array on the stack and memsetting this array with the marker value
- **Problem**: during execution the kernel stack is utilized an the marker bytes may be overwritten by subfunctions
- **Solution**: we utilize compiler instrumentation to re-taint
  - `-fsanitize-coverage=trace-pc`
  - compiler inserts call to `__sanitizer_cov_trace_pc`, where we employ the tainting but on a smaller scale
KLEAK
- Tainting Memory Sources (Stack)

```c
int SYS_SYSCALL(struct lwp *l, const struct CONCAT(SYS_SYSCALL, _args) *uap, register_t *rval)
{
    /*
    * {
    *       syscallarg(int) code;
    *       syscallarg(register_t) args[SYS_MAXSYSARGS];
    *    } */
    const struct sysent *callp;
    struct proc *p = l->l_proc;
    int code;
    int error;
    /* ...
    */
    callp = p->p_emul->e_sysent;

    code = SCARG(uap, code) & (SYS_NSYSENT - 1);
    SYSCALL_COUNT(syscall_counts, code);
    callp += code;
    /* ...
    */
    kleak_fill_stack();
    error = sy_call(callp, l, &uap->args, rval);
    /* ...
    */
    return error;
}
```
KLEAK
- Detecting Leaks at the Data Sinks

- We define \textit{copyout} and \textit{copyoutstr} as our data sinks
- On each invocation, we count the occurrences of the marker value
What are good values to use as markers?

Not all values are suitable, e.g. bytes 0 and 255

**Idea**: estimate byte frequency first and decide which bytes to use

- Environment: NetBSD 8.0 AMD64
- Patched `copyout` and `copyoutstr` to log copied bytes to data structures
- Added syscalls to fetch data from kernel space
- Ran NetBSD test suite `tests(7)`
KLEAK
- Choice of Marker Values 2/2

<table>
<thead>
<tr>
<th>rank</th>
<th>byte</th>
<th>ASCII</th>
<th>rank</th>
<th>byte</th>
<th>ASCII</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0</td>
<td>.</td>
<td>247</td>
<td>207</td>
<td>.</td>
</tr>
<tr>
<td>2</td>
<td>97</td>
<td>a</td>
<td>248</td>
<td>181</td>
<td>.</td>
</tr>
<tr>
<td>3</td>
<td>255</td>
<td>.</td>
<td>249</td>
<td>206</td>
<td>.</td>
</tr>
<tr>
<td>4</td>
<td>101</td>
<td>e</td>
<td>250</td>
<td>167</td>
<td>.</td>
</tr>
<tr>
<td>5</td>
<td>99</td>
<td>c</td>
<td>251</td>
<td>169</td>
<td>.</td>
</tr>
<tr>
<td>6</td>
<td>100</td>
<td>d</td>
<td>252</td>
<td>159</td>
<td>.</td>
</tr>
<tr>
<td>7</td>
<td>98</td>
<td>b</td>
<td>253</td>
<td>218</td>
<td>.</td>
</tr>
<tr>
<td>8</td>
<td>91</td>
<td>[</td>
<td>254</td>
<td>221</td>
<td>.</td>
</tr>
<tr>
<td>9</td>
<td>234</td>
<td>.</td>
<td>255</td>
<td>157</td>
<td>.</td>
</tr>
<tr>
<td>10</td>
<td>102</td>
<td>f</td>
<td>256</td>
<td>154</td>
<td>.</td>
</tr>
</tbody>
</table>
KLEAK
- Rotation of Marker Values

- Using only one marker byte results in a considerable amount of FP
- **Solution**: invoke kernel entrypoint in several rounds with changing marker bytes
- Implementation via *bitmap*
  - Each byte consists of 8 bits, each bit represents a round

```c
for (i = 0; i < n_rounds; i++) {
    invoke_kernel_entry_point ();
    if (i < n_rounds - 1)
        update_marker ();
}
dump_results ();
```
KLEAK
- Implementation in NetBSD

- Not enabled per default: developer option
- Userland tool `kleak` enables devs to check their syscalls

```
$ kleak -n 4 ps
[... output of ps ...]
Possible info leak: [len=1056, leaked=931]
#0 0xffffffff80bad351 in kleak_copyout
#1 0xffffffff80b2cf64 in uvm_swap_stats.part.1
#2 0xffffffff80b2d38d in uvm_swap_stats
#3 0xffffffff80b2d43c in sys_swapctl
#4 0xffffffff80259b82 in syscall
```
KLEAK - Limitations

- Simplicity and speed over precision
- Code coverage
- Portability
## KLEAK
- (Direct) Results on FreeBSD 11.2 and NetBSD-current (AMD64)

<table>
<thead>
<tr>
<th>OS</th>
<th>module</th>
<th>syscall</th>
<th>bytes leaked/copied</th>
<th>source</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>NetBSD</strong></td>
<td>sys/uvm/uvm_swap.c</td>
<td>swapctl</td>
<td>931/1056</td>
<td>kernel stack</td>
</tr>
<tr>
<td><strong>NetBSD</strong></td>
<td>sys/kern/sys_ptrace_common.c</td>
<td>ptrace</td>
<td>92/136</td>
<td>dynamic memory</td>
</tr>
<tr>
<td><strong>NetBSD</strong></td>
<td>sys/arch/amd64/amd64/machdep.c</td>
<td>signal</td>
<td>92/920</td>
<td>kernel stack</td>
</tr>
<tr>
<td><strong>NetBSD</strong></td>
<td>sys/kern/kern_time.c</td>
<td>settime50</td>
<td>16/32</td>
<td>dynamic memory</td>
</tr>
<tr>
<td><strong>NetBSD</strong></td>
<td>sys/kern/kern_exec.c</td>
<td>execve</td>
<td>8/32</td>
<td>kernel stack</td>
</tr>
<tr>
<td><strong>NetBSD</strong></td>
<td>sys/kern/kern_time.c</td>
<td>getitimer50</td>
<td>8/32</td>
<td>kernel stack</td>
</tr>
<tr>
<td><strong>FreeBSD</strong></td>
<td>sys/kern/sys_process.c</td>
<td>ptrace</td>
<td>8/176</td>
<td>kernel stack</td>
</tr>
<tr>
<td><strong>NetBSD</strong></td>
<td>sys/kern/kern_exit.c</td>
<td>wait6</td>
<td>4/4</td>
<td>kernel stack</td>
</tr>
<tr>
<td><strong>FreeBSD</strong></td>
<td>sys/ufs/ufs/ufs_vops.c</td>
<td>getdirent</td>
<td>4+/variable</td>
<td>kernel stack</td>
</tr>
<tr>
<td><strong>NetBSD</strong></td>
<td>sys/kern/kern_time.c</td>
<td>gettimeofday50</td>
<td>4/16</td>
<td>kernel stack</td>
</tr>
<tr>
<td><strong>NetBSD</strong></td>
<td>sys/kern/kern_sig.c</td>
<td>sigaction_sigtresp</td>
<td>4/32</td>
<td>dynamic memory</td>
</tr>
<tr>
<td><strong>FreeBSD</strong></td>
<td>sys/netinet/raw_ip.c</td>
<td>sysct1 (rip_pcblist)</td>
<td>4/32</td>
<td>kernel stack</td>
</tr>
<tr>
<td><strong>FreeBSD</strong></td>
<td>sys/netinet/tcp_subr.c</td>
<td>sysctl (tcp_pcblist)</td>
<td>4/32</td>
<td>kernel stack</td>
</tr>
<tr>
<td><strong>FreeBSD</strong></td>
<td>sys/netinet/udp_usrreq.c</td>
<td>sysctl (udp_pcblist)</td>
<td>4/32</td>
<td>kernel stack</td>
</tr>
<tr>
<td><strong>FreeBSD</strong></td>
<td>sys/kern/uipc_usrreq.c</td>
<td>sysctl (unp_pcblist)</td>
<td>4/32</td>
<td>dynamic memory</td>
</tr>
<tr>
<td><strong>NetBSD</strong></td>
<td>sys/kern/kern_event.c</td>
<td>kevent50</td>
<td>4/40</td>
<td>kernel stack</td>
</tr>
<tr>
<td><strong>NetBSD</strong></td>
<td>sys/kern/sys_sig.c</td>
<td>sigtimedwait50</td>
<td>4/40</td>
<td>kernel stack</td>
</tr>
<tr>
<td><strong>FreeBSD</strong></td>
<td>sys/kern/kern_ntptime.c</td>
<td>sysctl (ntp_sysctl)</td>
<td>4/48</td>
<td>kernel stack</td>
</tr>
<tr>
<td><strong>NetBSD</strong></td>
<td>sys/kern/kern_ntptime.c</td>
<td>ntp_gettime</td>
<td>4/48</td>
<td>kernel stack</td>
</tr>
<tr>
<td><strong>FreeBSD</strong></td>
<td>sys/net/rtsock.c</td>
<td>sysctl (rsock)</td>
<td>4/232</td>
<td>dynamic memory</td>
</tr>
<tr>
<td><strong>NetBSD</strong></td>
<td>sys/net/rtsock.c</td>
<td>sysctl (rtable)</td>
<td>2/176</td>
<td>dynamic memory</td>
</tr>
</tbody>
</table>
# KLEAK

- (Direct) Results on FreeBSD 11.2 and NetBSD-current (AMD64)

<table>
<thead>
<tr>
<th>OS</th>
<th>module</th>
<th>syscall</th>
<th>bytes leaked/copied</th>
<th>source</th>
</tr>
</thead>
<tbody>
<tr>
<td>NetBSD</td>
<td>sys/uvm/uvm_swap.c</td>
<td>swapctl</td>
<td>931/1056</td>
<td>kernel stack</td>
</tr>
<tr>
<td>NetBSD</td>
<td>sys/kern/sys_ptrace_common.c</td>
<td>ptrace</td>
<td>92/136</td>
<td>dynamic memory</td>
</tr>
<tr>
<td>NetBSD</td>
<td>sys/arch/amd64/amd64/machdep</td>
<td>signal</td>
<td>92/920</td>
<td>kernel stack</td>
</tr>
<tr>
<td>NetBSD</td>
<td>sys/kern/kern_time.c</td>
<td>settime50</td>
<td>16/32</td>
<td>dynamic memory</td>
</tr>
<tr>
<td>NetBSD</td>
<td>sys/kern/kern_exec.c</td>
<td>execve</td>
<td>8/32</td>
<td>kernel stack</td>
</tr>
<tr>
<td>NetBSD</td>
<td>sys/kern/kern_time.c</td>
<td>gettimer50</td>
<td>8/32</td>
<td>kernel stack</td>
</tr>
<tr>
<td>FreeBSD</td>
<td>sys/kern/sys_process.c</td>
<td>ptrace</td>
<td>8/176</td>
<td>kernel stack</td>
</tr>
<tr>
<td>NetBSD</td>
<td>sys/kern/kern_exit.c</td>
<td>wait6</td>
<td>4/4</td>
<td>kernel stack</td>
</tr>
<tr>
<td>FreeBSD</td>
<td>sys/uufs/uufs_ups.c</td>
<td>getdirent</td>
<td>4/32</td>
<td>kernel stack</td>
</tr>
<tr>
<td>FreeBSD</td>
<td>sys/kern/kern_time.c</td>
<td>settime50</td>
<td>8/32</td>
<td>kernel stack</td>
</tr>
<tr>
<td>FreeBSD</td>
<td>sys/kern/kern_start.c</td>
<td>sys action</td>
<td>8/32</td>
<td>dynamic memory</td>
</tr>
<tr>
<td>FreeBSD</td>
<td>sys/netinet/raw_ip.c</td>
<td>sysctl (rip_pcblist)</td>
<td>4/32</td>
<td>kernel stack</td>
</tr>
<tr>
<td>FreeBSD</td>
<td>sys/netinet/tcp_subr.c</td>
<td>sysctl (tcp_pcblist)</td>
<td>4/32</td>
<td>kernel stack</td>
</tr>
<tr>
<td>FreeBSD</td>
<td>sys/netinet/udp_usrreq.c</td>
<td>sysctl (udp_pcblist)</td>
<td>4/32</td>
<td>kernel stack</td>
</tr>
<tr>
<td>FreeBSD</td>
<td>sys/kern/ufp_usreq.c</td>
<td>sysctl (unp_pcblist)</td>
<td>4/32</td>
<td>dynamic memory</td>
</tr>
<tr>
<td>NetBSD</td>
<td>sys/kern/kern_event.c</td>
<td>kevent50</td>
<td>4/40</td>
<td>kernel stack</td>
</tr>
<tr>
<td>NetBSD</td>
<td>sys/kern/sys_sig.c</td>
<td>sigtimedwait50</td>
<td>4/40</td>
<td>kernel stack</td>
</tr>
<tr>
<td>FreeBSD</td>
<td>sys/kern/kern_ntptime.c</td>
<td>sysctl (ntp_sysctl)</td>
<td>4/48</td>
<td>kernel stack</td>
</tr>
<tr>
<td>NetBSD</td>
<td>sys/kern/kern_ntptime.c</td>
<td>ntp_gettime</td>
<td>4/48</td>
<td>kernel stack</td>
</tr>
<tr>
<td>FreeBSD</td>
<td>sys/net/rtsock.c</td>
<td>sysctl (rtsock)</td>
<td>4/232</td>
<td>dynamic memory</td>
</tr>
<tr>
<td>NetBSD</td>
<td>sys/net/rtsock.c</td>
<td>sysctl (rttable)</td>
<td>2/176</td>
<td>dynamic memory</td>
</tr>
</tbody>
</table>
### FreeBSD/src 340856 — head/sys/cddl/contrib/openSolaris/uts/common/fs/zfs/ctldir.c

**mark@head** — 2018-11-23 22:24:59 UTC

Ensure that directory entry padding bytes are zeroed.

Directory entries must be padded to maintain alignment in many filesystems. The padding was not initialized, resulting in stack memory being copied out to user space. With the ino64 work there are also some explicit pad fields in struct dirent. Add a subroutine to clear these bytes and use it in the in-tree filesystems. The NFT client is emitted for now as it was fixed separately in r340787.

**Reported by:** Thomas Barabosch, Fraunhofer FKIE
**Reviewed by:** kib
**MFC after:** 3 days
**Sponsored by:** The FreeBSD Foundation

<table>
<thead>
<tr>
<th>#</th>
<th>Change Request</th>
<th>Summary</th>
</tr>
</thead>
<tbody>
<tr>
<td>+13</td>
<td>-0</td>
<td>head/sys/syse/dirent.h</td>
</tr>
<tr>
<td>+4</td>
<td>-7</td>
<td>head/sys/fs/autofs/autofs_vnops.c</td>
</tr>
<tr>
<td>+4</td>
<td>-7</td>
<td>head/sys/fs/tmpfs/tmpfs_subr.c</td>
</tr>
<tr>
<td>+2</td>
<td>-2</td>
<td>head/sys/cddl/contrib/openSolaris/uts/common/fs/zfs/ctldir.c</td>
</tr>
<tr>
<td>+2</td>
<td>-2</td>
<td>head/sys/fs/mdofsdfs/mdofsdfs_vnops.c</td>
</tr>
<tr>
<td>+3</td>
<td>-3</td>
<td>head/sys/fs/nandfs/nandfs_vnops.c</td>
</tr>
<tr>
<td>+3</td>
<td>-3</td>
<td>head/sys/fs/udfs/udfs_vnops.c</td>
</tr>
<tr>
<td>+2</td>
<td>-2</td>
<td>head/sys/fs/smbfs/smbfs_lo.c</td>
</tr>
<tr>
<td>+1</td>
<td>-1</td>
<td>head/sys/fs/fsdevfs/fsdevfs_vnops.c</td>
</tr>
<tr>
<td>+1</td>
<td>-1</td>
<td>head/sys/fs/ufs/ufs_vnops.c</td>
</tr>
<tr>
<td>+1</td>
<td>-1</td>
<td>head/sys/fs/fuse/fuse_internal.c</td>
</tr>
<tr>
<td>+1</td>
<td>-1</td>
<td>head/sys/fs/pseudo/pseudo_vnops.c</td>
</tr>
<tr>
<td>+1</td>
<td>-1</td>
<td>head/sys/fs/tmpfs/tmpfs_vnops.c</td>
</tr>
<tr>
<td>+1</td>
<td>-1</td>
<td>head/sys/fs/tmpfs/tmpfs_vnops.c</td>
</tr>
<tr>
<td>+1</td>
<td>-1</td>
<td>head/sys/kernel/uipc rhyme.c</td>
</tr>
<tr>
<td>+1</td>
<td>-1</td>
<td>head/sys/kernel/vfs_export.c</td>
</tr>
<tr>
<td>+1</td>
<td>-1</td>
<td>head/sys/cddl/contrib/openSolaris/uts/common/fs/zfs/vnops.c</td>
</tr>
<tr>
<td>+50</td>
<td>-33</td>
<td>20 files</td>
</tr>
</tbody>
</table>

### FreeBSD/src 340787 — head/sys/fs/nfsclient/nfs_clrcpops.c

**rmacklem@head** — 2018-11-23 00:17:47 UTC

Make sure the NFS readdir client fills all “struct dirent” data.

The NFS client code (nfscl_readdir() and nfscl_readdirplus()) wasn’t filling in parts of the readdir reply, such as d_pad[0] and the bytes at the end of d_name when d_reclen. As such, data left in a buffer cache block could be leaked to userland in the readdir reply. This patch makes sure all of the data is filled in.

**Reported by:** Thomas Barabosch, Fraunhofer FKIE
**Reviewed by:** kib, markj
**MFC after:** 2 weeks

<table>
<thead>
<tr>
<th>#</th>
<th>Change Request</th>
<th>Summary</th>
</tr>
</thead>
<tbody>
<tr>
<td>+14</td>
<td>-8</td>
<td>head/sys/fs/nfsclient/nfs_clrcpops.c</td>
</tr>
<tr>
<td>+14</td>
<td>-8</td>
<td>1 files</td>
</tr>
</tbody>
</table>

### FreeBSD/src 340699 — head/sys/kern/kern_ntptime.c

**mark@head** — 2018-11-20 20:32:10 UTC

Clear pad bytes in the struct exported by kern.ntp.pll.gettime.

**Reported by:** Thomas Barabosch, Fraunhofer FKIE
**Reviewed by:** kib
**MFC after:** 3 days

<table>
<thead>
<tr>
<th>#</th>
<th>Change Request</th>
<th>Summary</th>
</tr>
</thead>
<tbody>
<tr>
<td>+2</td>
<td>-0</td>
<td>head/sys/kern/kern_ntptime.c</td>
</tr>
<tr>
<td>+2</td>
<td>-0</td>
<td>1 files</td>
</tr>
</tbody>
</table>

### FreeBSD/src 340783 — head/sys/kern/uipc_socket.c uipc_usreq.c, head/sys/netinet/tcp_subr.c in pcb.c

**mark@head** — 2018-11-22 20:49:41 UTC

Plug some networking syscall leaks.

Various network protocol syscall handlers were not zero-filling their output buffers and thus would export uninitialized stack memory to userland. Fix a number of such handlers.

**Reported by:** Thomas Barabosch, Fraunhofer FKIE
**Reviewed by:** sjenkins
**MFC after:** 3 days
**Security:** kernel memory disclosure
**Sponsored by:** The FreeBSD Foundation

<table>
<thead>
<tr>
<th>#</th>
<th>Change Request</th>
<th>Summary</th>
</tr>
</thead>
<tbody>
<tr>
<td>+50</td>
<td>-33</td>
<td>20 files</td>
</tr>
</tbody>
</table>

### FreeBSD/src 344142 — head/sys/amd64/amd64 machdep.c, head/sys/amd64/amd64 machdep.c

**mark@head** — 2018-12-03 20:54:17 UTC

Plug memory disclosures via ptzio2().

On some architectures, the structures returned by PT_GETHESS were not fully populated and could contain uninitialized stack memory. The same issue existed with the register files in proofs.

**Reported by:** Thomas Barabosch, Fraunhofer FKIE
**Reviewed by:** sjenkins
**MFC after:** 3 days
**Security:** kernel stack memory disclosure
**Sponsored by:** The FreeBSD Foundation

<table>
<thead>
<tr>
<th>#</th>
<th>Change Request</th>
<th>Summary</th>
</tr>
</thead>
<tbody>
<tr>
<td>+13</td>
<td>-3</td>
<td>head/sys/kern/sys_process.c</td>
</tr>
<tr>
<td>+3</td>
<td>-2</td>
<td>head/sys/fs/procfs/procfs_fwpages.c</td>
</tr>
<tr>
<td>+3</td>
<td>-2</td>
<td>The FreeBSD Foundation</td>
</tr>
<tr>
<td>+2</td>
<td>-1</td>
<td>head/sys/net/inet/ipdiv.c</td>
</tr>
<tr>
<td>+1</td>
<td>-1</td>
<td>head/sys/net/inet/ipredirect.c</td>
</tr>
<tr>
<td>+1</td>
<td>-1</td>
<td>head/sys/net/inet/ipc Pan.c</td>
</tr>
<tr>
<td>+1</td>
<td>-1</td>
<td>head/sys/net/inet/uipc_usreq.c</td>
</tr>
<tr>
<td>+1</td>
<td>-1</td>
<td>head/sys/ufs/ufs_vnops.c</td>
</tr>
<tr>
<td>+1</td>
<td>-1</td>
<td>head/sys/ufs/ufs_vnops.c</td>
</tr>
<tr>
<td>+1</td>
<td>-1</td>
<td>head/sys/ufs/ufs_vnops.c</td>
</tr>
<tr>
<td>+10</td>
<td>-8</td>
<td>10 files</td>
</tr>
</tbody>
</table>

### FreeBSD/src 340968 — head/sys/net rtsock.c if.h

**mark@head** — 2018-11-26 13:42:18 UTC

Plug routing syscall leaks.

Various structures exported by nsysctlsockopt() contain padding fields which were not being zeroed.

**Reported by:** Thomas Barabosch, Fraunhofer FKIE
**Reviewed by:** sjenkins
**MFC after:** 3 days
**Security:** kernel memory disclosure

<table>
<thead>
<tr>
<th>#</th>
<th>Change Request</th>
<th>Summary</th>
</tr>
</thead>
<tbody>
<tr>
<td>+14</td>
<td>-2</td>
<td>head/sys/net/rtsock.c</td>
</tr>
<tr>
<td>+4</td>
<td>-0</td>
<td>head/sys/net/if.c</td>
</tr>
<tr>
<td>+19</td>
<td>-2</td>
<td>head/sys/net/rtsock.c</td>
</tr>
</tbody>
</table>
Conclusion

- Kernel Memory Disclosures – what they are and how to avoid them
- KLEAK detected more than 20 KMDs in NetBSD–current/FreeBSD 11.2
  - dozens of KMDs were manually detected as a follow-up
- KLEAK fully integrated in NetBSD –current; FreeBSD port is still missing
- **One more thing**: the *BSDs are far from being KMD-free!