KLEAK

Practical Kernel Memory Disclosure Detection

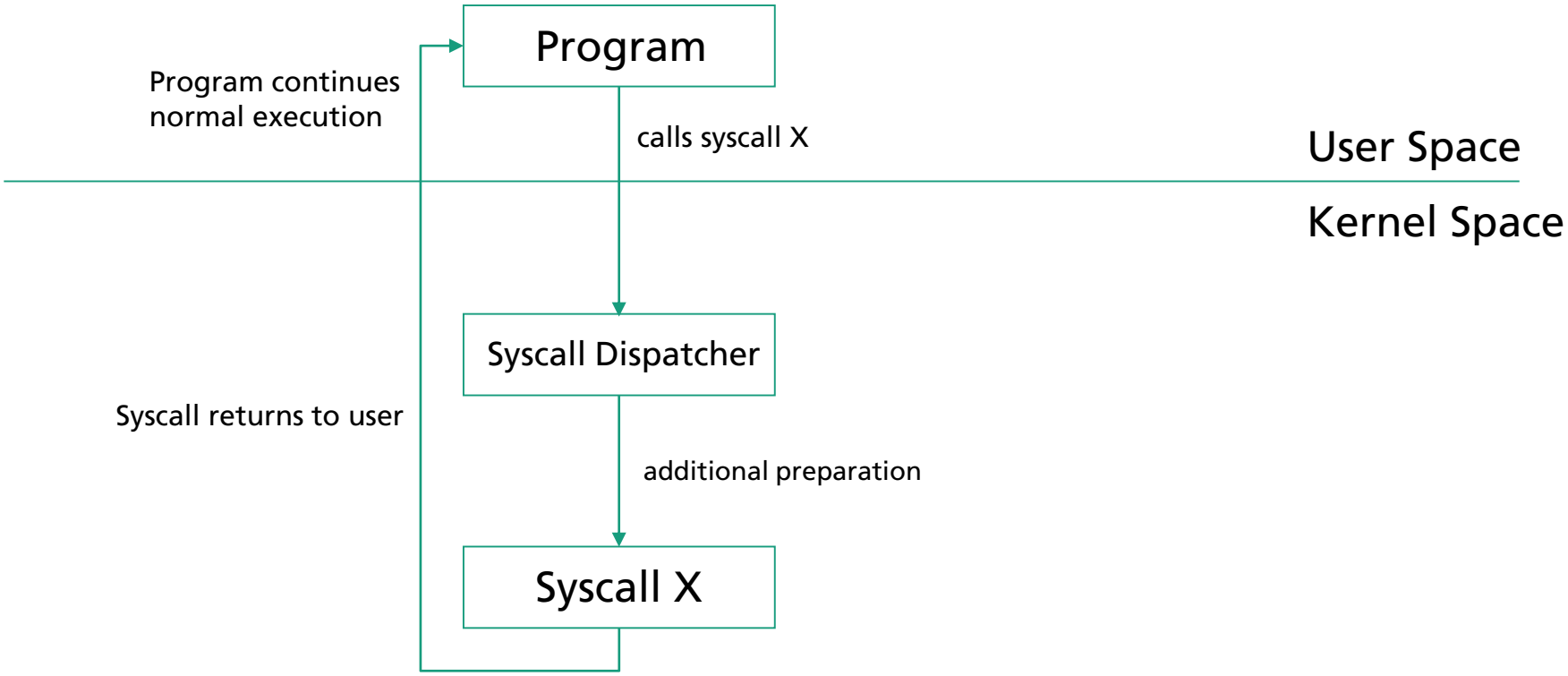
Fraunhofer
FKIE

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

Maxime Villard
NetBSD
m00nbsd.net

System Call

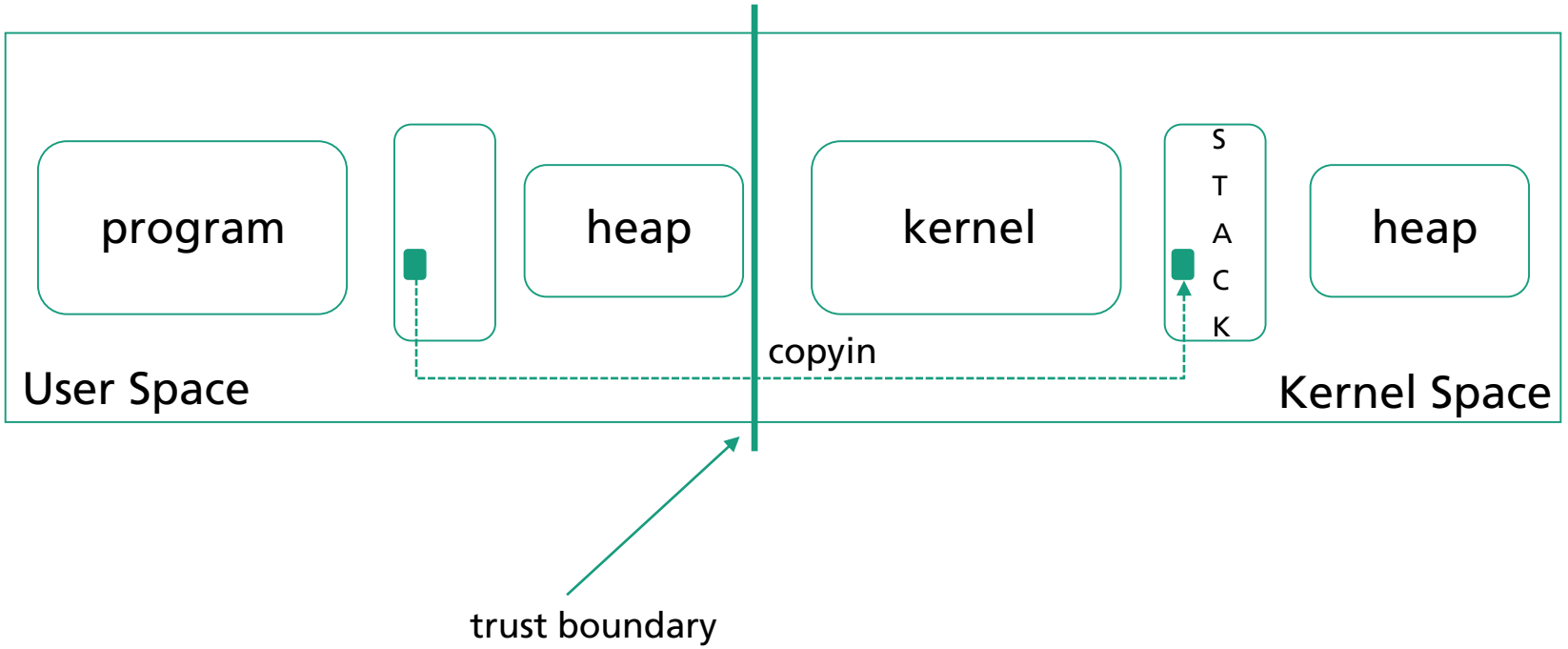
- Recap



Kernel/User Space Data Exchange

- copyin

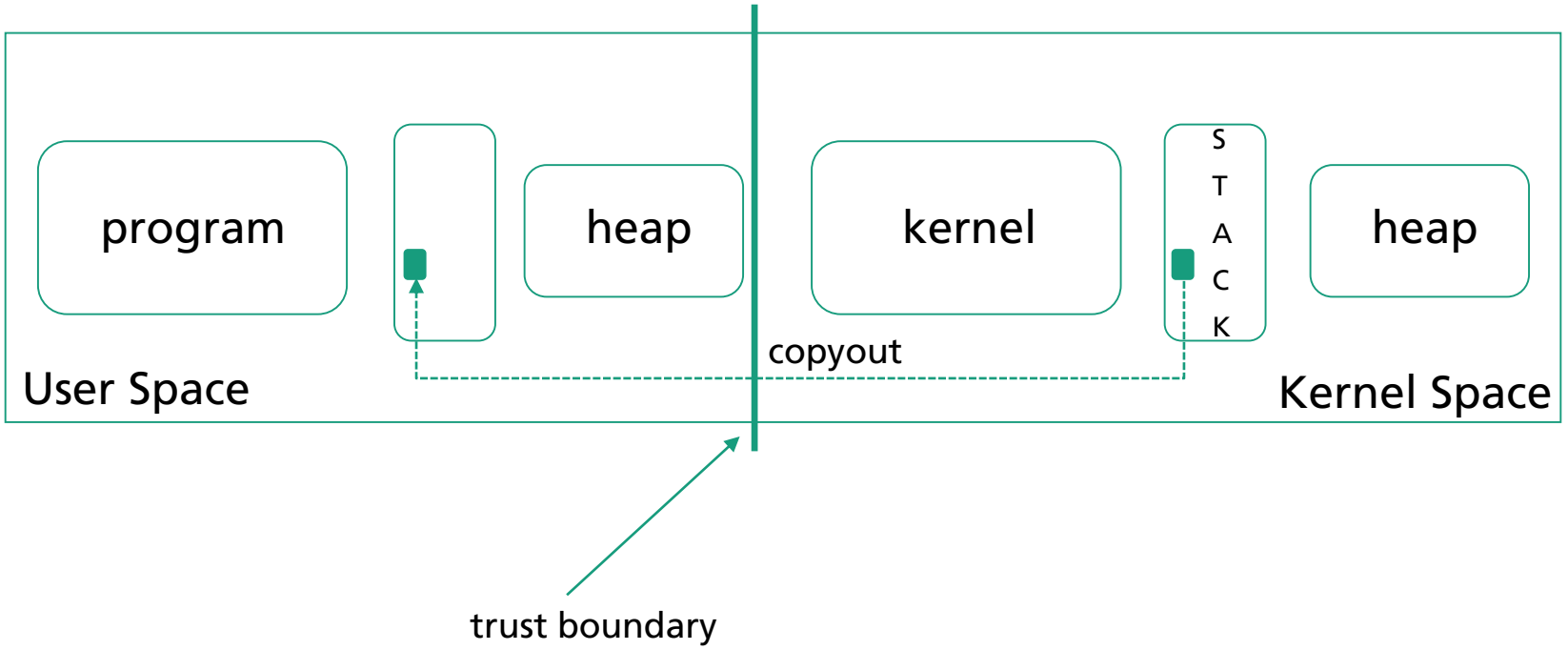
- User space programs can not directly write to kernel space
- It points the kernel to a buffer, the kernel fetches this data
  - copyin, copyinstr, …



Kernel/User Space Data Exchange

- copyout

- Kernel uses dedicated functions to copy data to user space
  - copyout, copyoutstr, …
- Supervisor Mode Access Prevention (SMAP)



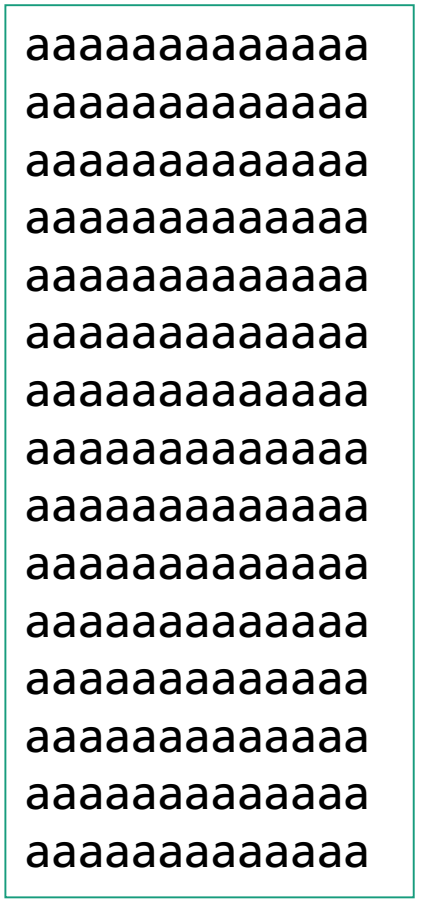
Kernel Memory Disclosure (KMD)
- What is it?

- 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!

Kernel Memory Disclosure

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

```
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);
}
```

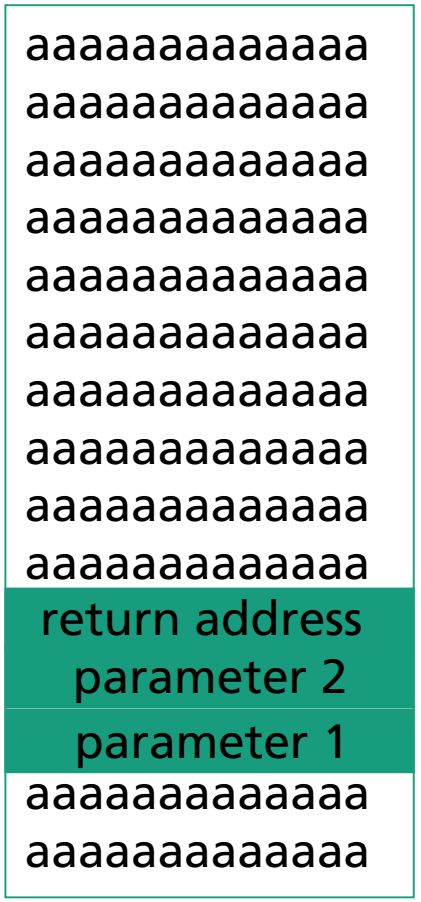


Stack

Kernel Memory Disclosure

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

```
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);
}
```

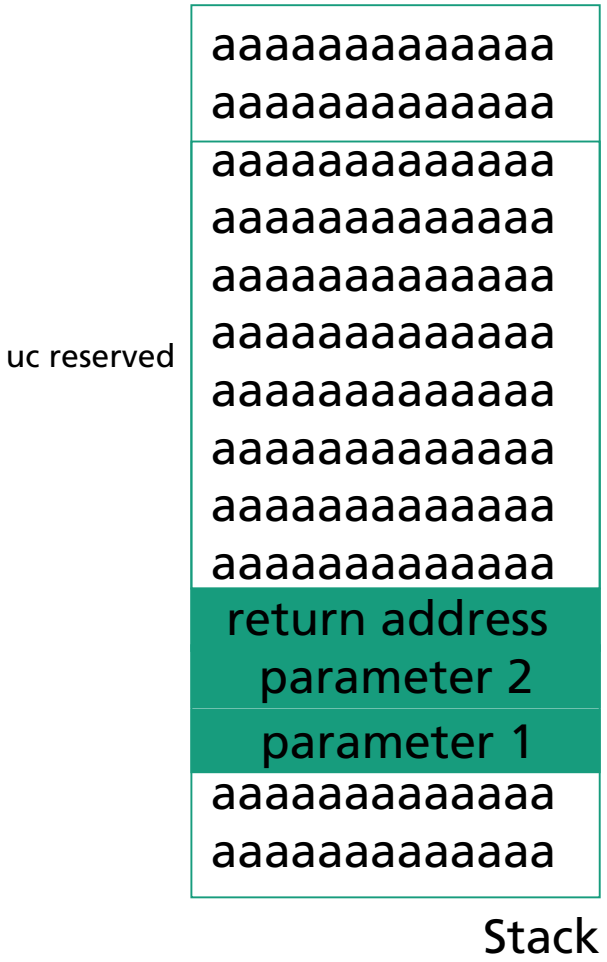


Stack

Kernel Memory Disclosure

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

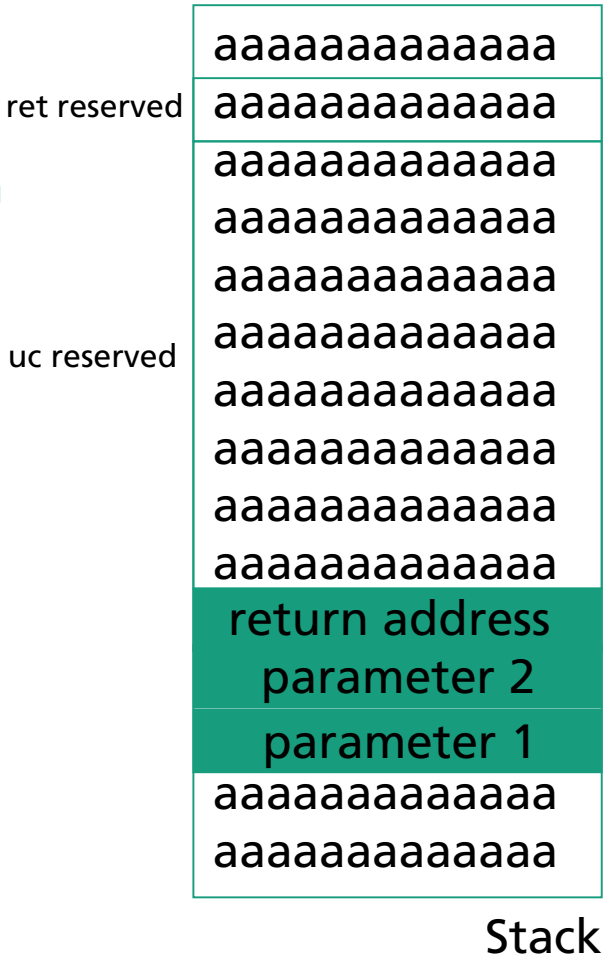
```
int sys_getcontext(struct thread *td,
                   struct getcontext_args *uap) {
    ucontext_t uc;
    int ret;

    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

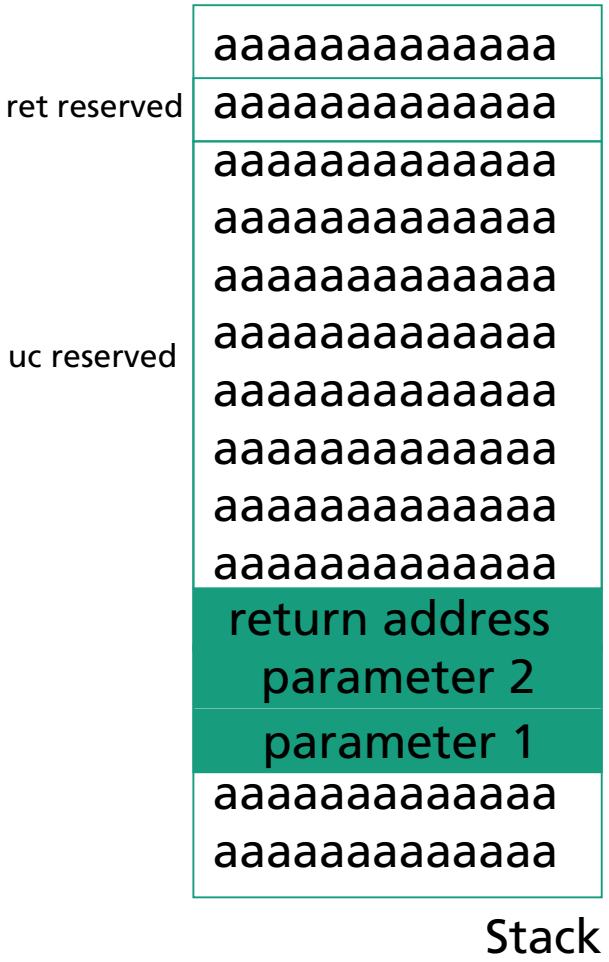
```
int sys_getcontext(struct thread *td,
                   struct getcontext_args *uap) {
    ucontext_t uc;
    int ret;

    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

```
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

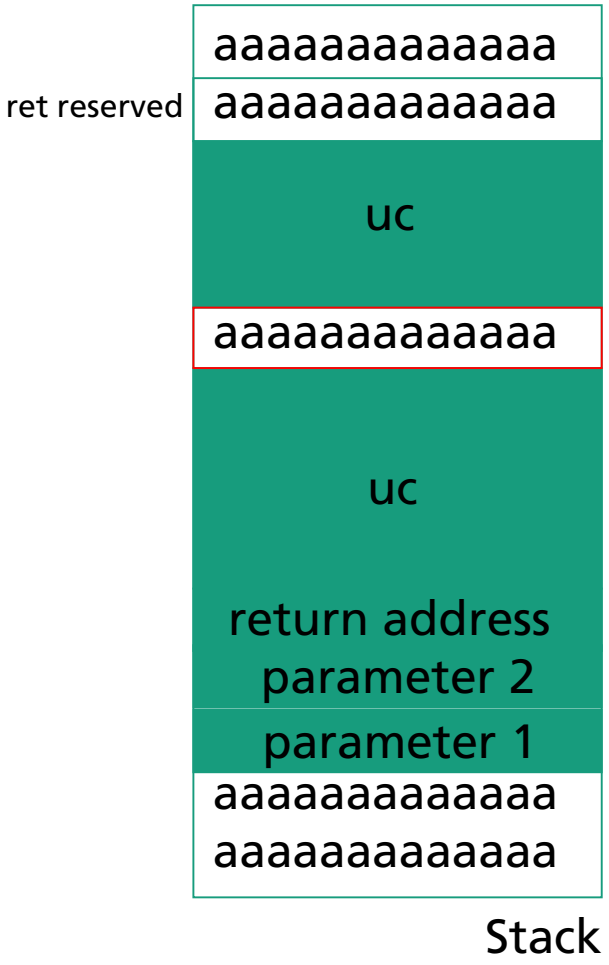
- CVE-2018-17155 / FreeBSD-EN-18:12.mem

```
struct ucontext4 {
    sigset_t    uc_sigmask;
    struct mcontext4 uc_mcontext;
    struct ucontext4 *uc_link;
    stack_t     uc_stack;
    int     __spare__[8];
};
```

Kernel Memory Disclosure

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

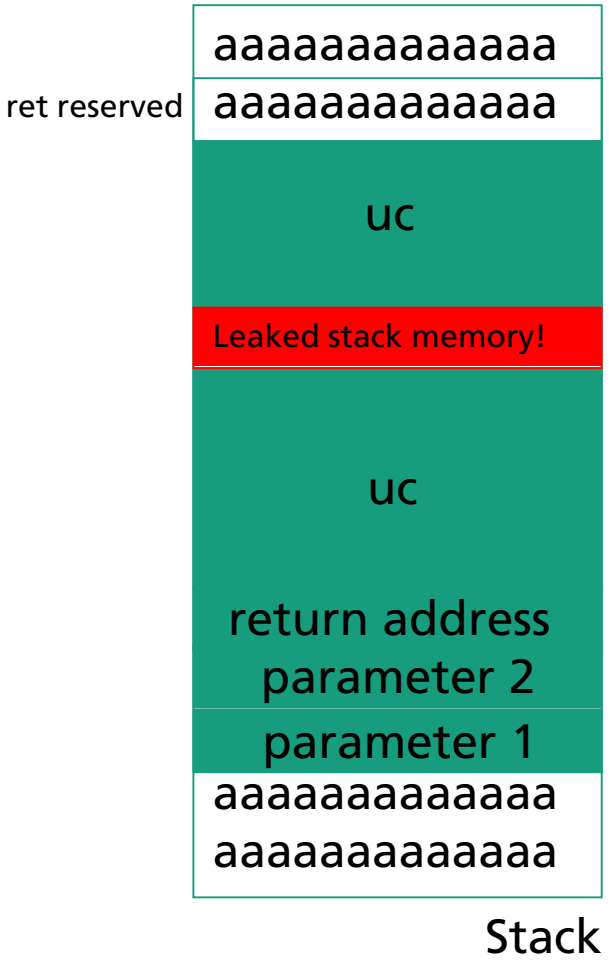
```
int sys_getcontext(struct thread *td,
                   struct getcontext_args *uap) {
    ucontext_t uc;
    int ret;

    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

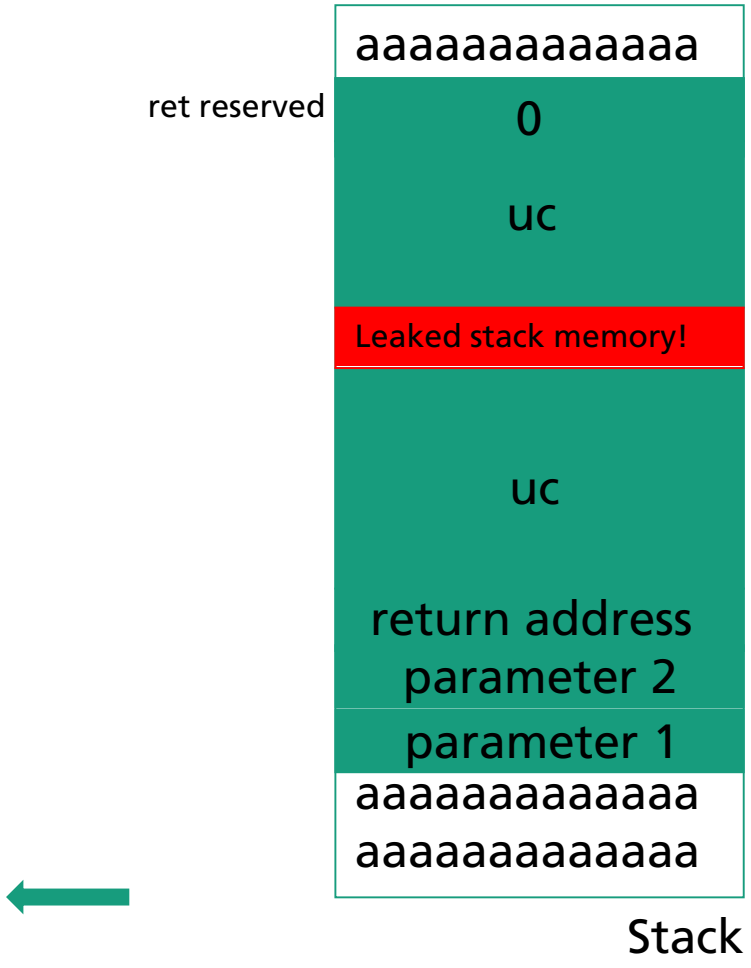
```
int sys_getcontext(struct thread *td,
                   struct getcontext_args *uap) {
    ucontext_t uc;
    int ret;

    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

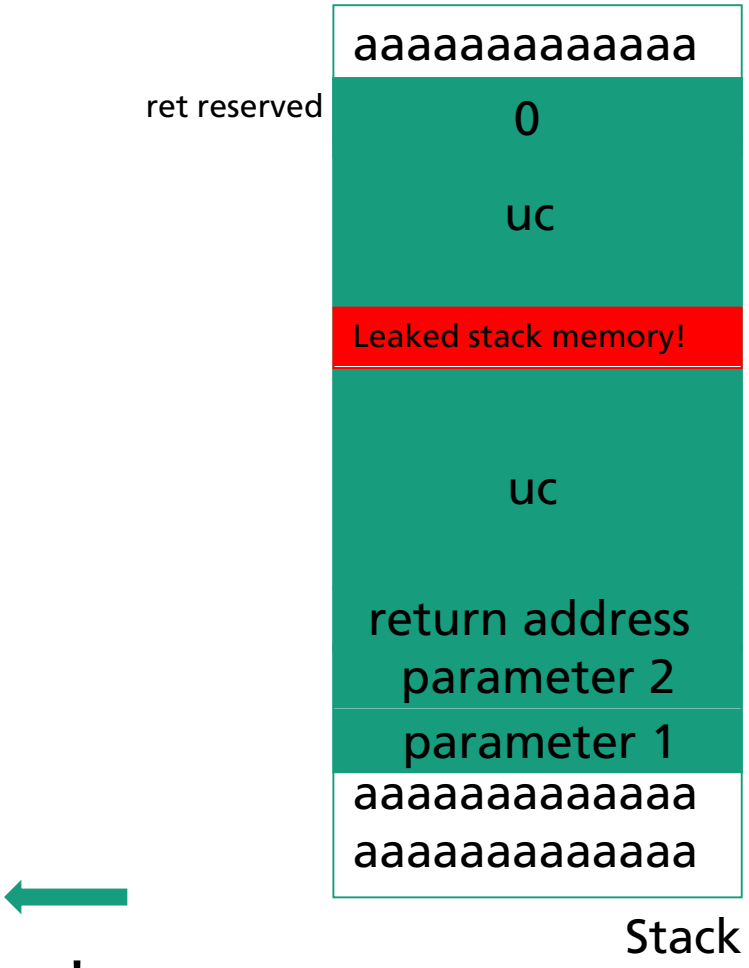
- CVE-2018-17155 / FreeBSD-EN-18:12.mem

```
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

```
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!

Kernel Memory Disclosure

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

```
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

Kernel Memory Disclosure
- How to avoid it? 1/2

- Remember that
  - local variables (stack) are unitialized
  - your heap implementation may return unitialized 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?

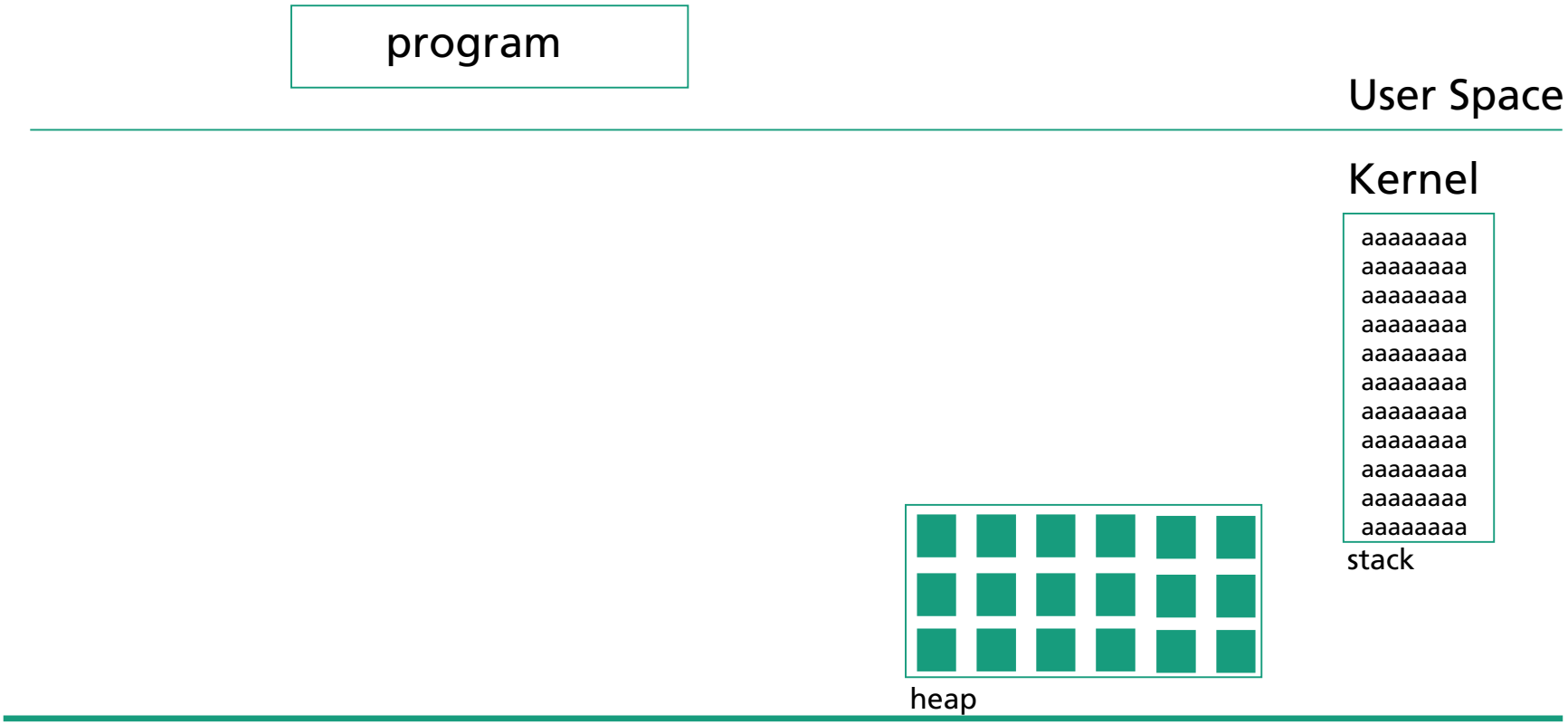
- Several researchers uncovered hundreds of KMDs
  - Lu et al. state that 37 KMDs were found in Linux in 2010/2011
  - Lu et al. found 19 KMDs in Linux and Android in 2016
  - Mateusz Jurczyk found 80 KMDs in Windows and Linux in 2018
  - …
- So far no systematic investigation on the *BSDs
  - Exception: 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

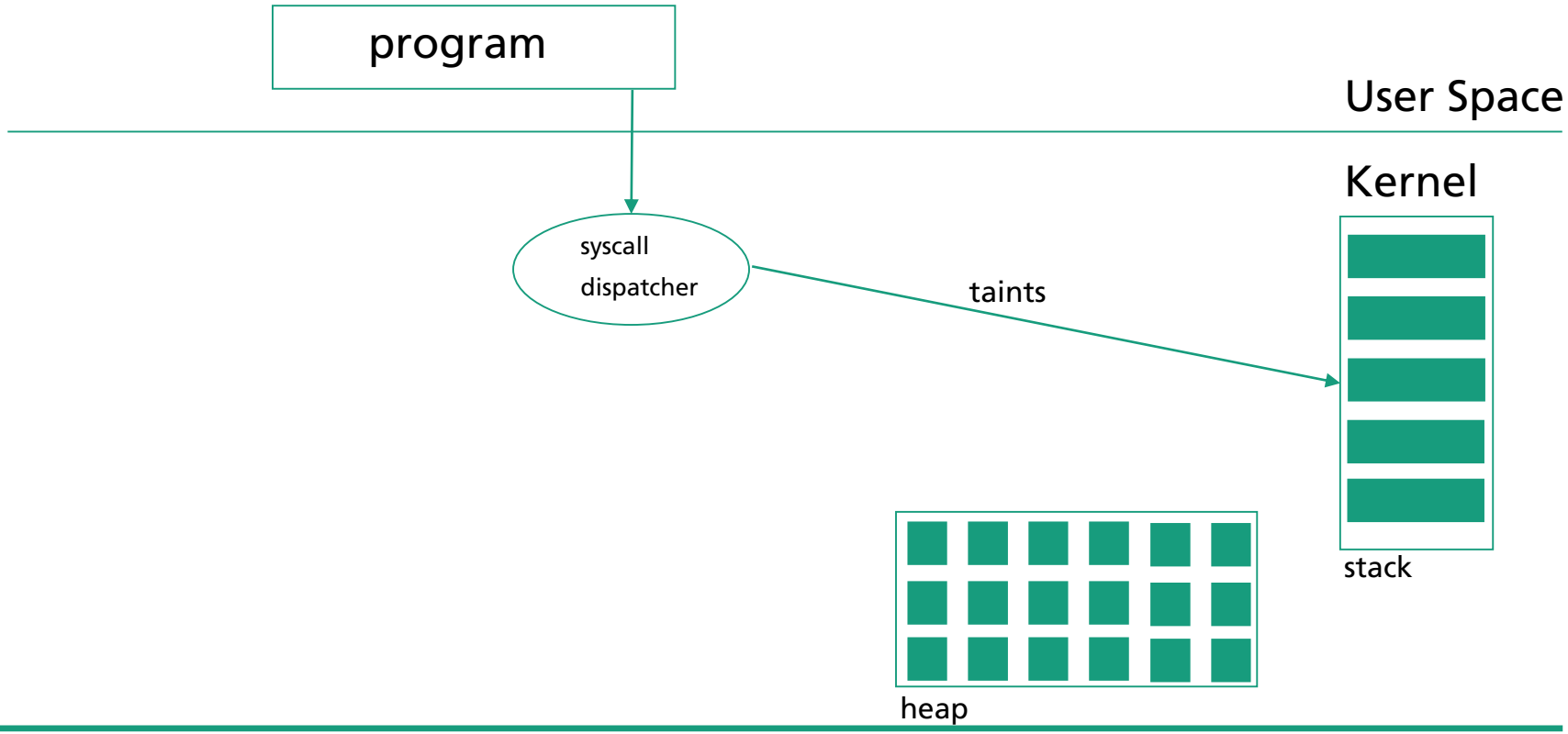
User Space

Kernel

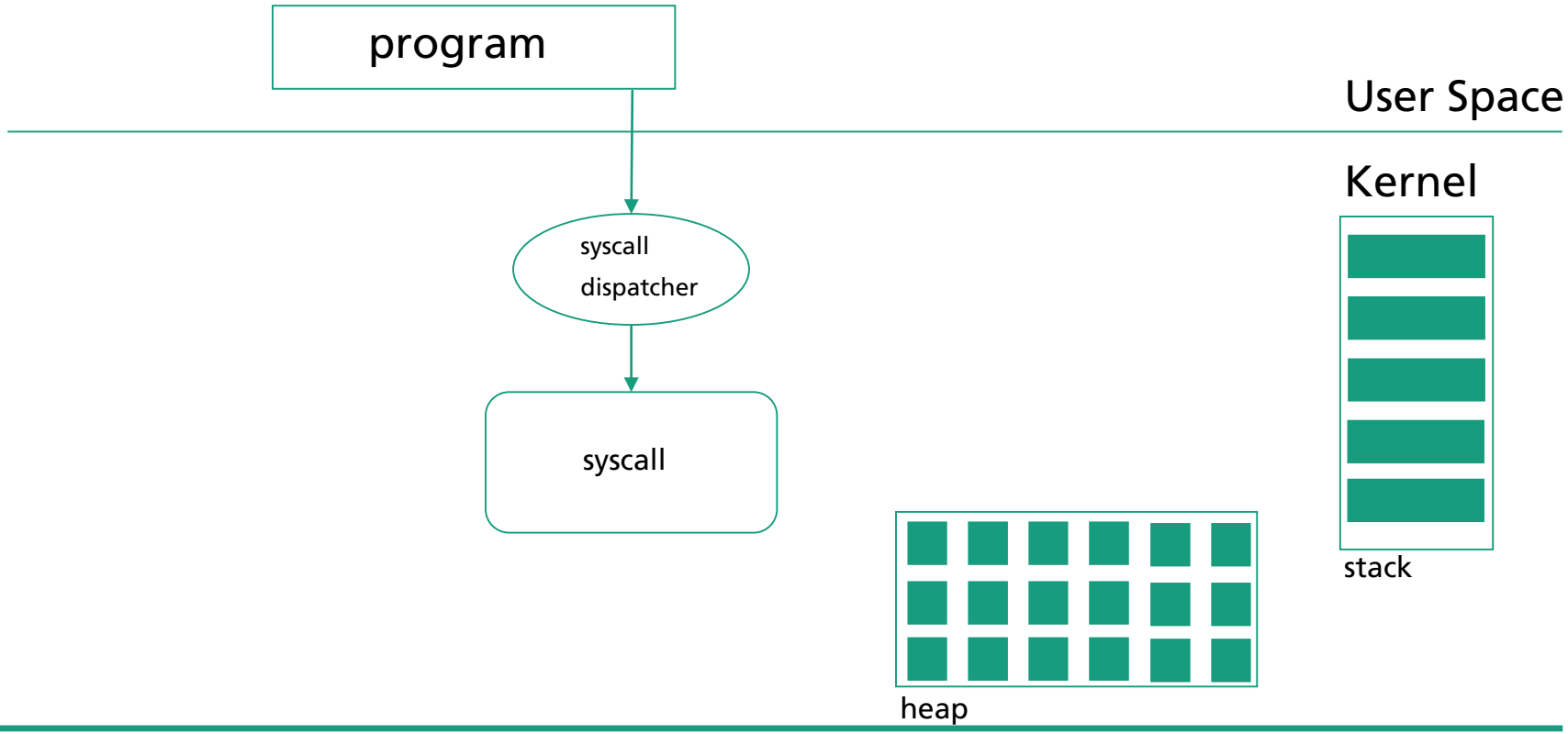
KLEAK
- Overview



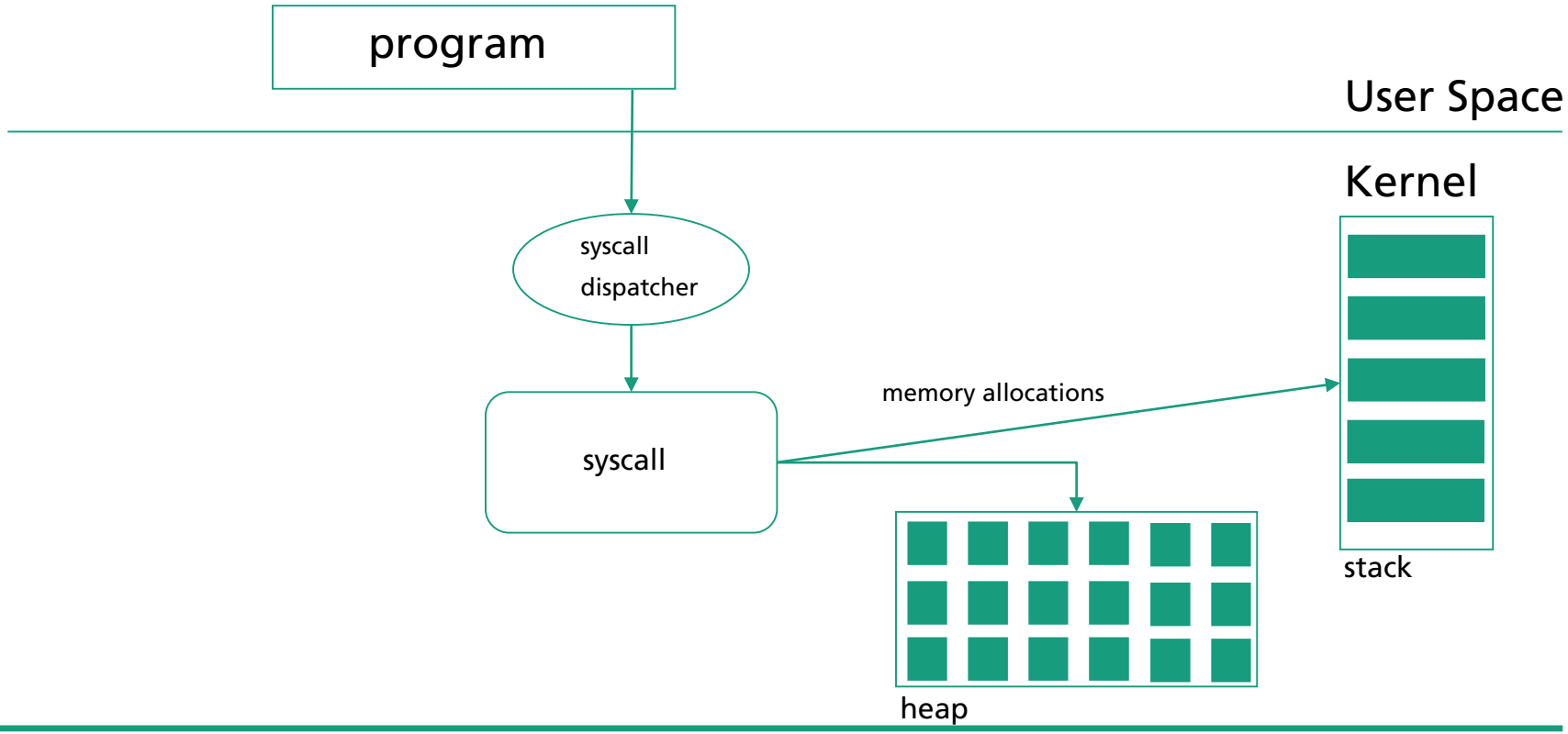
KLEAK
- Overview



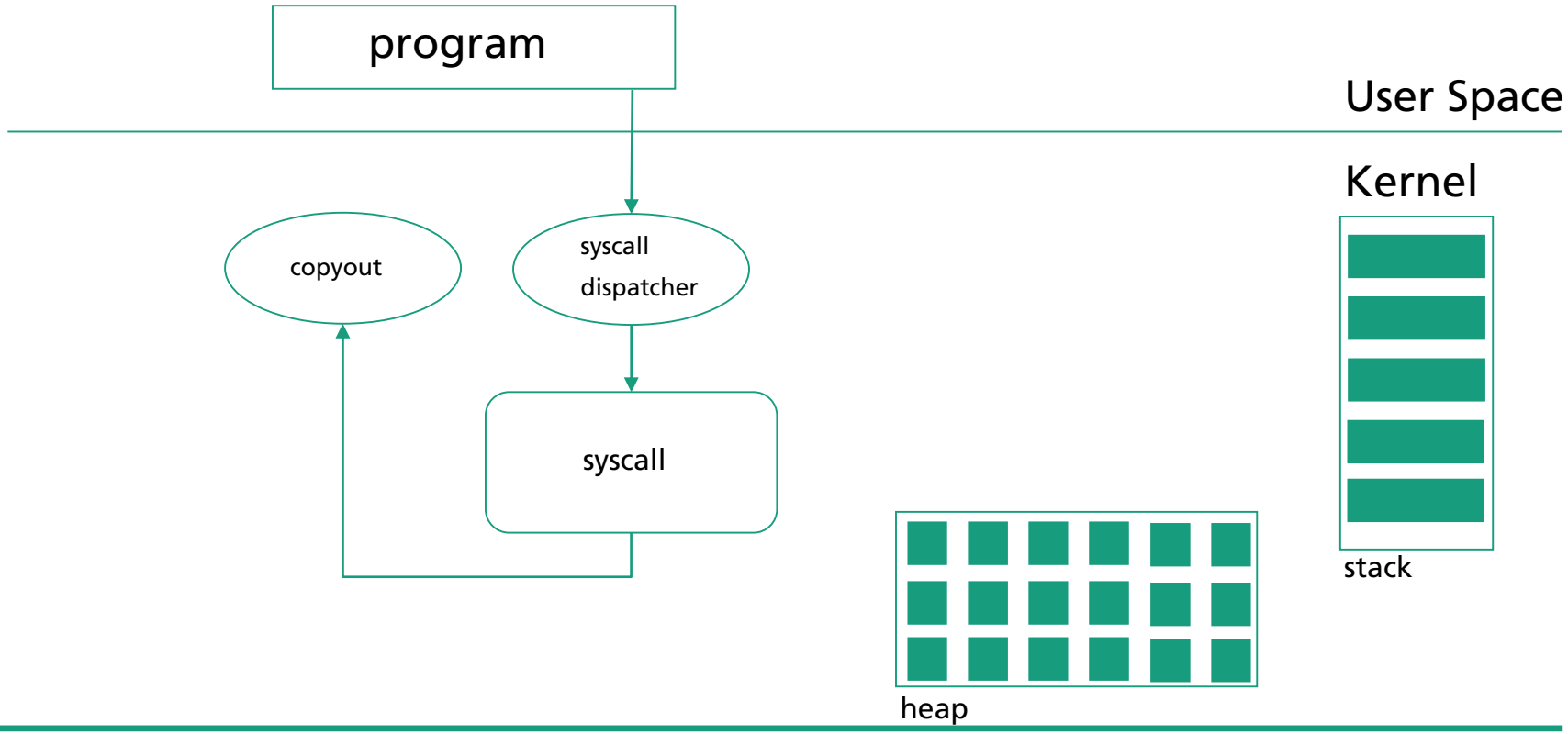
KLEAK
- Overview



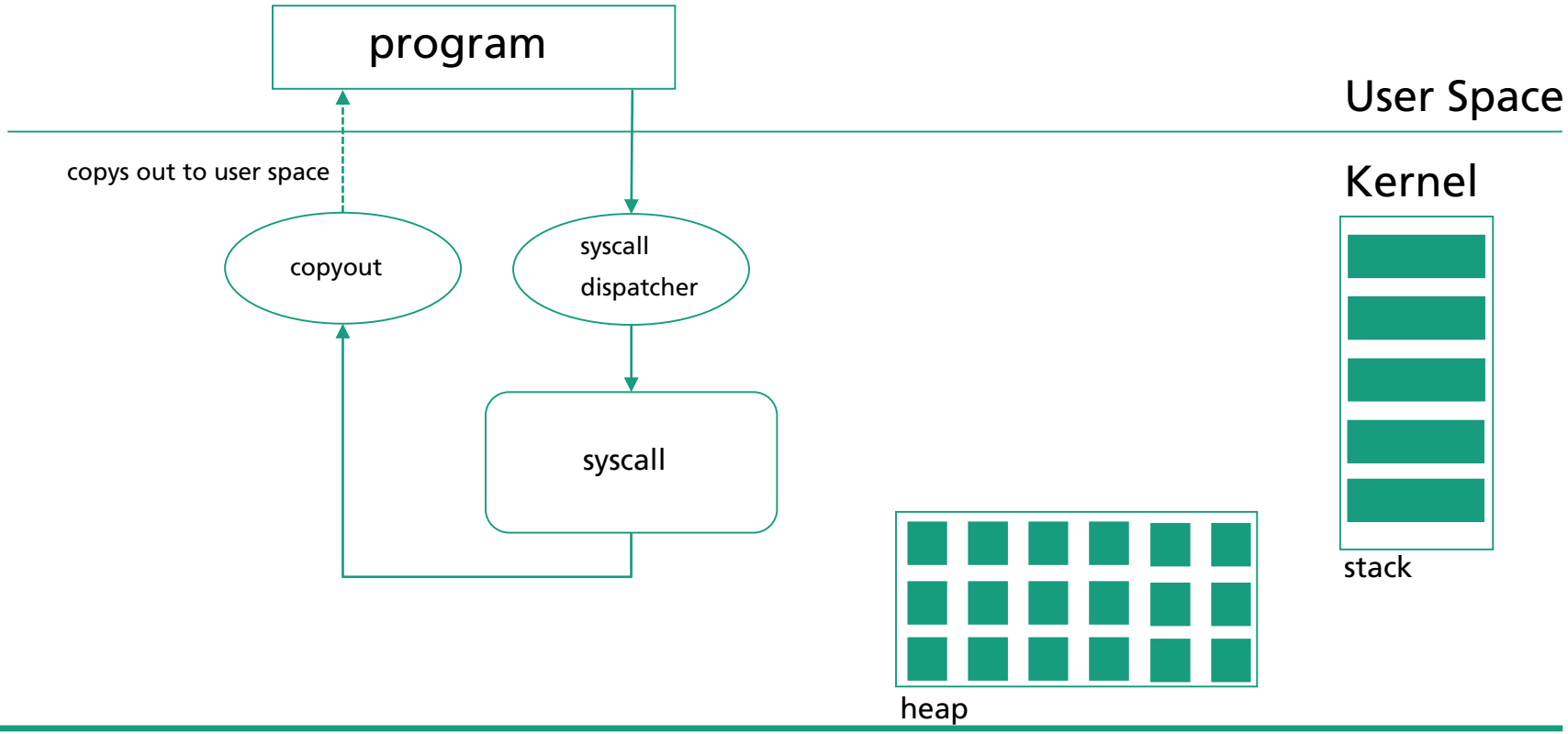
KLEAK
- Overview



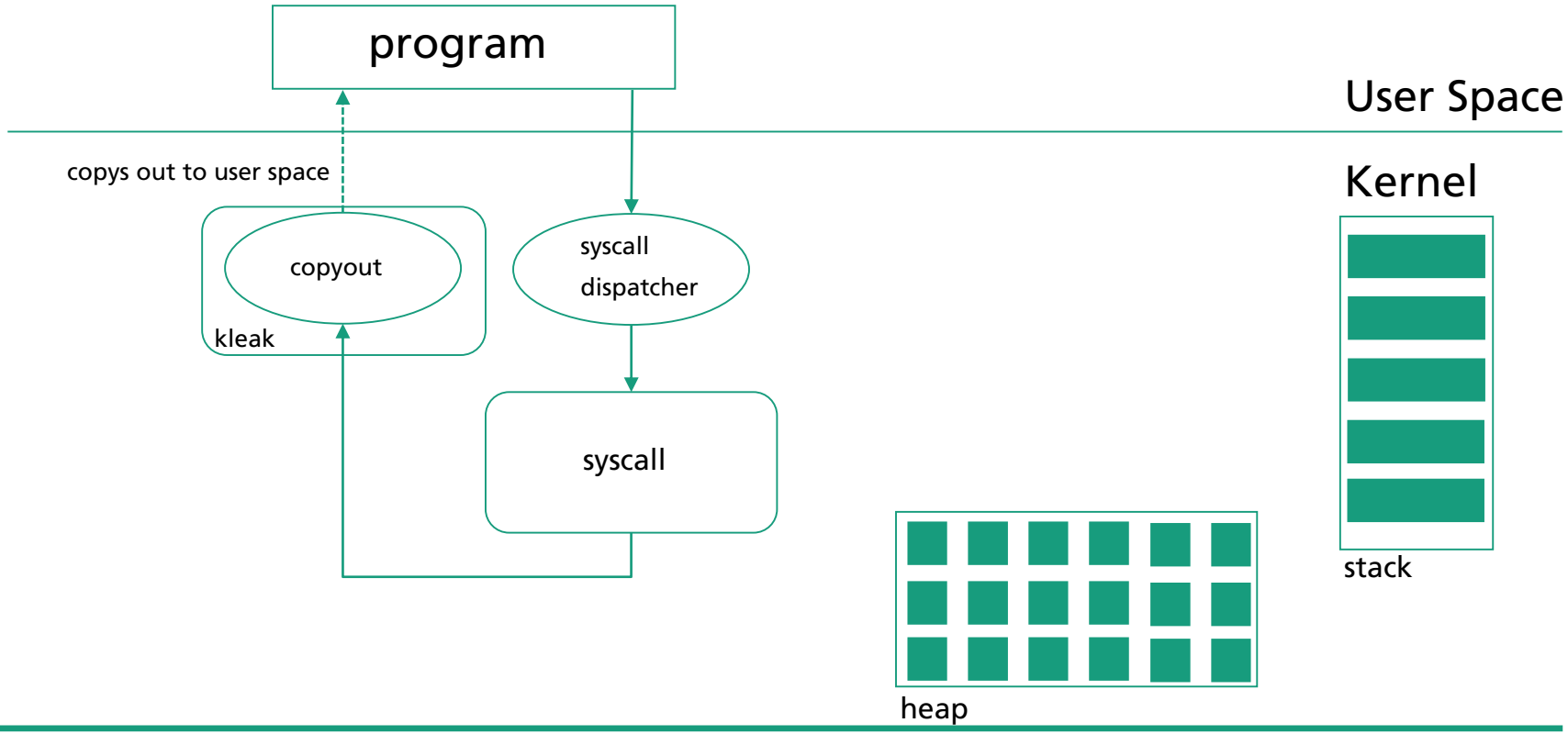
KLEAK
- Overview



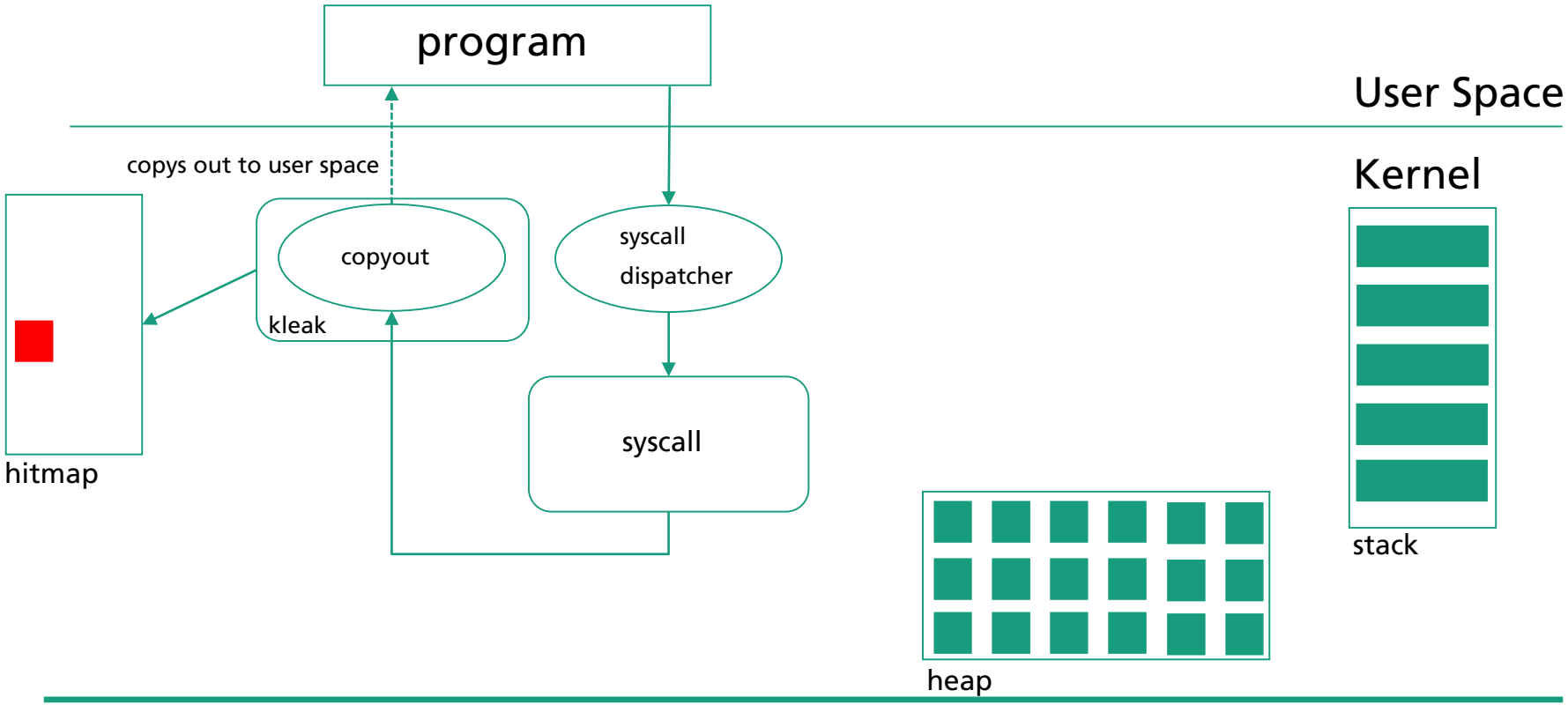
KLEAK
- Overview



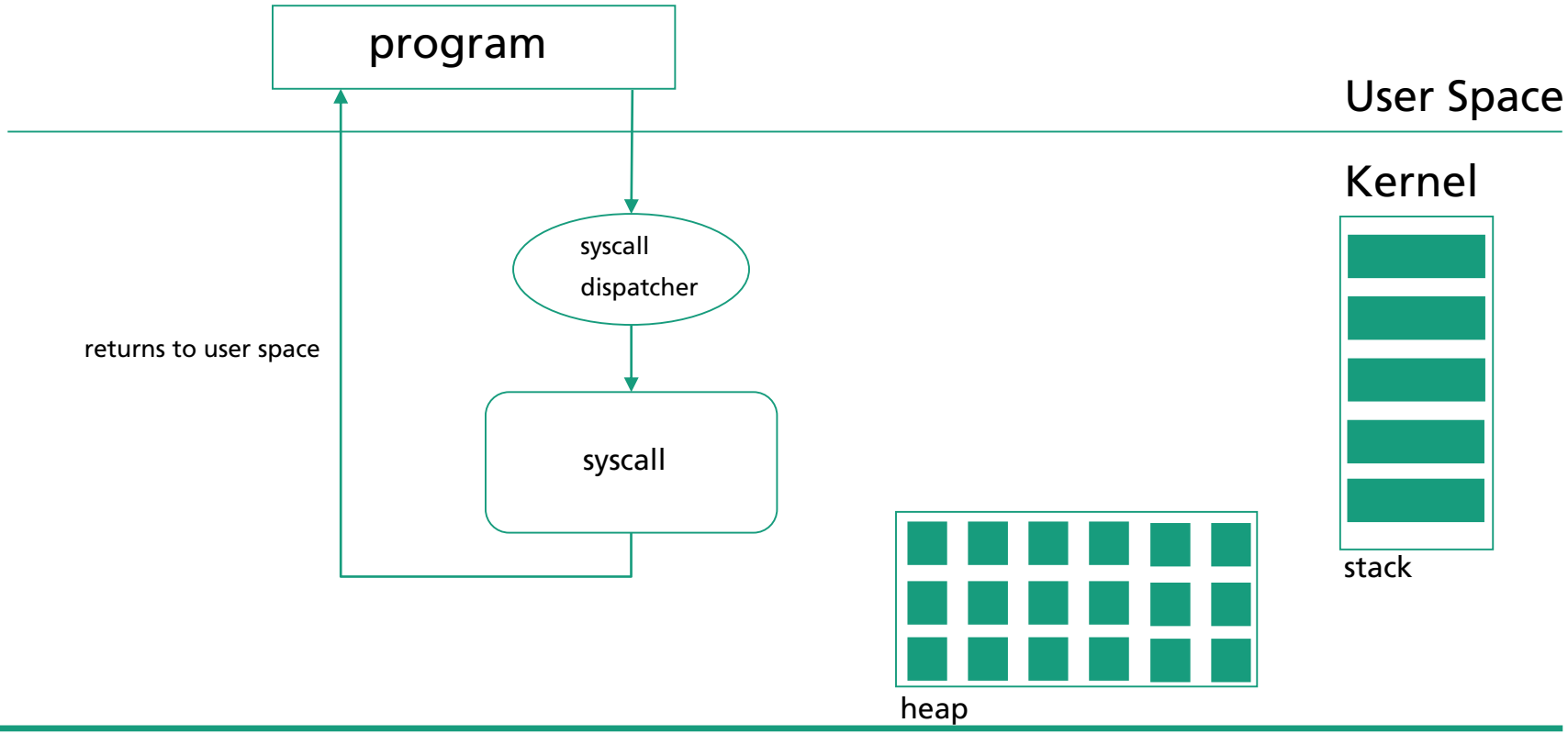
KLEAK
- Overview



KLEAK
- Overview



KLEAK
- Overview



KLEAK
- 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)

```
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

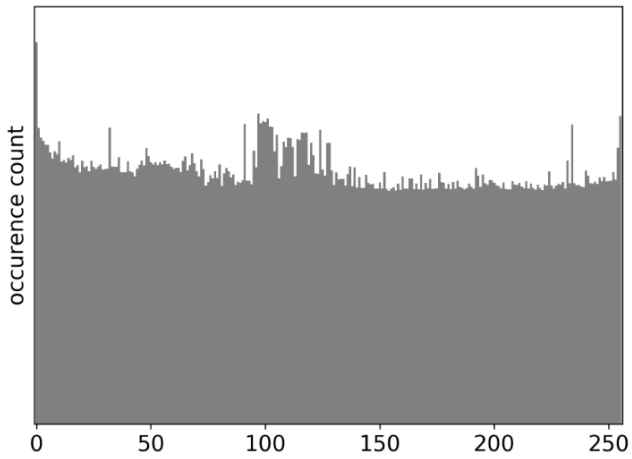
- Detecting Leaks at the Data Sinks

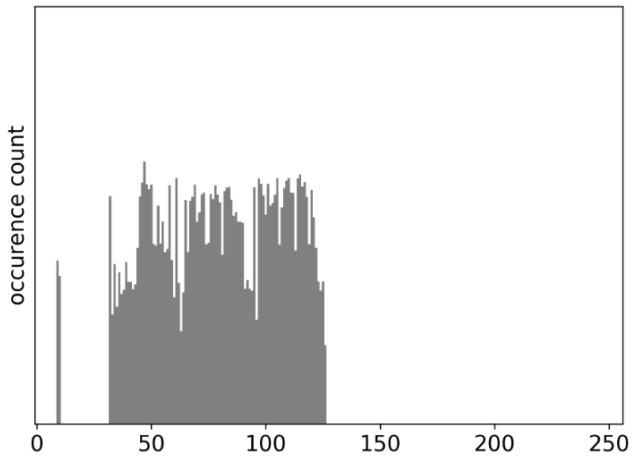
- We define *copyout* and *copyoutstr* as our data sinks
- On each invokation, we count the occurences of the marker value

KLEAK
- Choice of Marker Values 1/2

- 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





rank	byte	ASCII	rank	byte	ASCII
1	0	.	247	207	.
2	97	a	248	181	.
3	255	.	249	206	.
4	101	e	250	167	.
5	99	c	251	169	.
6	100	d	252	159	.
7	98	b	253	218	.
8	91	[	254	221	.
9	234	.	255	157	.
10	102	f	256	154	.

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 *hitmap*
  - Each byte consists of 8 bits, each bit represents a round

```
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)

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

KLEAK

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

OS	module	syscall	bytes leaked/copied	source
NetBSD	sys/uvm/uvm_swap.c	swapctl	931/1056	kernel stack
NetBSD	sys/kern/sys_ptrace_common.c	ptrace	92/136	dynamic memory
NetBSD	sys/arch/amd64/amd64/machdep.c	signal	92/920	kernel stack
NetBSD	sys/kern/kern_time.c	settime50	16/32	dynamic memory
NetBSD	sys/kern/kern_exec.c	execve	8/32	kernel stack
NetBSD	sys/kern/kern_time.c	getitimer50	8/32	kernel stack
FreeBSD	sys/kern/sys_process.c	ptrace	8/176	kernel stack
NetBSD	sys/kern/kern_exit.c	wait6	4/4	kernel stack
FreeBSD	sys/ufs/[illegible]	[illegible]	4/[illegible]	kernel stack
NetBSD	sys/kern/kern_[illegible]	[illegible]	[illegible]	kernel [illegible]
NetBSD	sys/kern/kern_sig.c	[illegible]	4/32	dynamic memory
FreeBSD	sys/netinet/raw_ip.c	sysctl (rip_pcblist)	4/32	kernel stack
FreeBSD	sys/netinet/tcp_subr.c	sysctl (tcp_pcblist)	4/32	kernel stack
FreeBSD	sys/netinet/udp_usrreq.c	sysctl (udp_pcblist)	4/32	kernel stack
FreeBSD	sys/kern/uipc_usrreq.c	sysctl (unp_pcblist)	4/32	dynamic memory
NetBSD	sys/kern/kern_event.c	kevent50	4/40	kernel stack
NetBSD	sys/kern/sys_sig.c	sigtimedwait50	4/40	kernel stack
FreeBSD	sys/kern/kern_ntptime.c	sysctl (ntp_sysctl)	4/48	kernel stack
NetBSD	sys/kern/kern_ntptime.c	ntp_gettime	4/48	kernel stack
FreeBSD	sys/net/rtsock.c	sysctl (rtsock)	4/232	dynamic memory
NetBSD	sys/net/rtsock.c	sysctl (rtable)	2/176	dynamic memory

KLEAK
- Follow-Up

FreeBSD/src 340856 — head/sys/cddl/contrib/opensolaris/uts/common/fs/zfs zfs_ctldir.c

markj@ 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 userspace.  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
NFS client is omitted 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
```

+13	-0	head/sys/sys/dirent.h
+4	-7	head/sys/fs/autofs/autofs_vnops.c
+4	-4	head/sys/fs/tmpfs/tmpfs_subr.c
+4	-2	head/sys/cddl/contrib/opensolaris/uts/common/fs/zfs/zfs_ctldir.c
+4	-2	head/sys/fs/msdosfs/msdosfs_vnops.c
+3	-3	head/sys/fs/nandfs/nandfs_vnops.c
+3	-2	head/sys/fs/udf/udf_vnops.c
+2	-2	head/sys/fs/smbfs/smbfs_io.c
+2	-1	head/sys/fs/fdescfs/fdesc_vnops.c
+1	-1	head/sys/fs/cd9660/cd9660_vnops.c
+1	-1	head/sys/fs/devfs/devfs_devs.c
+1	-1	head/sys/fs/ext2fs/ext2_lookup.c
+1	-1	head/sys/fs/fuse/fuse_internal.c
+1	-1	head/sys/fs/pseudofs/pseudofs_vnops.c
+1	-1	head/sys/fs/tmpfs/tmpfs_vfsops.c
+1	-1	head/sys/fs/tmpfs/tmpfs_vnops.c
+1	-1	head/sys/kern/uipc_mqueue.c
+1	-1	head/sys/kern/vfs_export.c
+1	-1	head/sys/ufs/ufs/ufs_vnops.c
+1	-0	head/sys/cddl/contrib/opensolaris/uts/common/fs/zfs/zfs_vnops.c
+50	-33	20 files

FreeBSD/src 340787 — head/sys/fs/nfsclient nfs_clrpcops.c

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

```
Make sure the NFS readdir client fills in all "struct dirent" data.

The NFS client code (nfsrpc_readdir() and nfsrpc_readdirplus()) wasn't
filling in parts of the readdir reply, such as d_pad[01] and the bytes
at the end of d_name within 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
```

+14	-8	head/sys/fs/nfsclient/nfs_clrpcops.c
+14	-8	1 files

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

markj@ 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
MFC after:      3 days
Sponsored by:   The FreeBSD Foundation
```

+2	-0	head/sys/kern/kern_ntptime.c
+2	-0	1 files

FreeBSD/src 340783 — head/sys/kern uipc_socket.c uipc_usrreq.c, head/sys/netinet tcp_subr.c in_pcb.c

markj@ head — 2018-11-22 20:49:41 UTC

```
Plug some networking sysctl leaks.

Various network protocol sysctl 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:    tuexen
MFC after:      3 days
Security:       kernel memory disclosure
Sponsored by:   The FreeBSD Foundation
Differential Revision:  https://reviews.freebsd.org/D18301
```

+3	-2	head/sys/netinet/tcp_subr.c
+1	-2	head/sys/kern/uipc_socket.c
+1	-2	head/sys/netinet/in_pcb.c
+3	-0	head/sys/netinet/sctp_sysctl.c
+2	-1	head/sys/netinet6/ip6_mroute.c
+1	-1	head/sys/kern/uipc_usrreq.c
+1	-0	head/sys/netinet/ip_divert.c
+1	-0	head/sys/netinet/raw_ip.c
+1	-0	head/sys/netinet/udp_usrreq.c
+1	-0	head/sys/ofed/drivers/infiniband/ulp/sdp/sdp_main.c
+15	-8	10 files

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

markj@ head — 2018-11-26 13:42:18 UTC

```
Plug routing sysctl leaks.

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

Reported by:    Thomas Barabosch, Fraunhofer FKIE
Reviewed by:    ae
MFC after:      3 days
Security:       kernel memory disclosure
Sponsored by:   The FreeBSD Foundation
Differential Revision:  https://reviews.freebsd.org/D18333
```

+14	-2	head/sys/net/rtsock.c
+4	-0	head/sys/net/if.h
+1	-0	head/sys/net/route.h
+19	-2	3 files

FreeBSD/src 341442 — head/sys/amd64/amd64 machdep.c, head/sys/amd6

markj@ head — 2018-12-03 20:54:17 UTC

```
Plug memory disclosures via ptrace(2).

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

Reported by:    Thomas Barabosch, Fraunhofer FKIE
Reviewed by:    kib
MFC after:      3 days
Security:       kernel stack memory disclosure
Sponsored by:   The FreeBSD Foundation
Differential Revision:  https://reviews.freebsd.org/D18421
```

+13	-3	head/sys/kern/sys_process.c
+3	-2	head/sys/fs/procfs/procfs_fpregs.c
+3	-2	head/sys/fs/procfs/procfs_regs.c
+4	-0	head/sys/arm/arm/machdep_kdb.c
+3	-1	head/sys/fs/procfs/procfs_dbregs.c
+3	-0	head/sys/amd64/amd64/machdep.c
+3	-0	head/sys/i386/i386/machdep.c
+2	-0	head/sys/amd64/ia32/ia32_reg.c
+1	-0	head/sys/sparc64/sparc64/machdep.c
+35	-8	9 files

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!