

K-Miner

Uncovering Memory Corruption in Linux

David Gens

Simon Schmitt

Ahmad-Reza Sadeghi

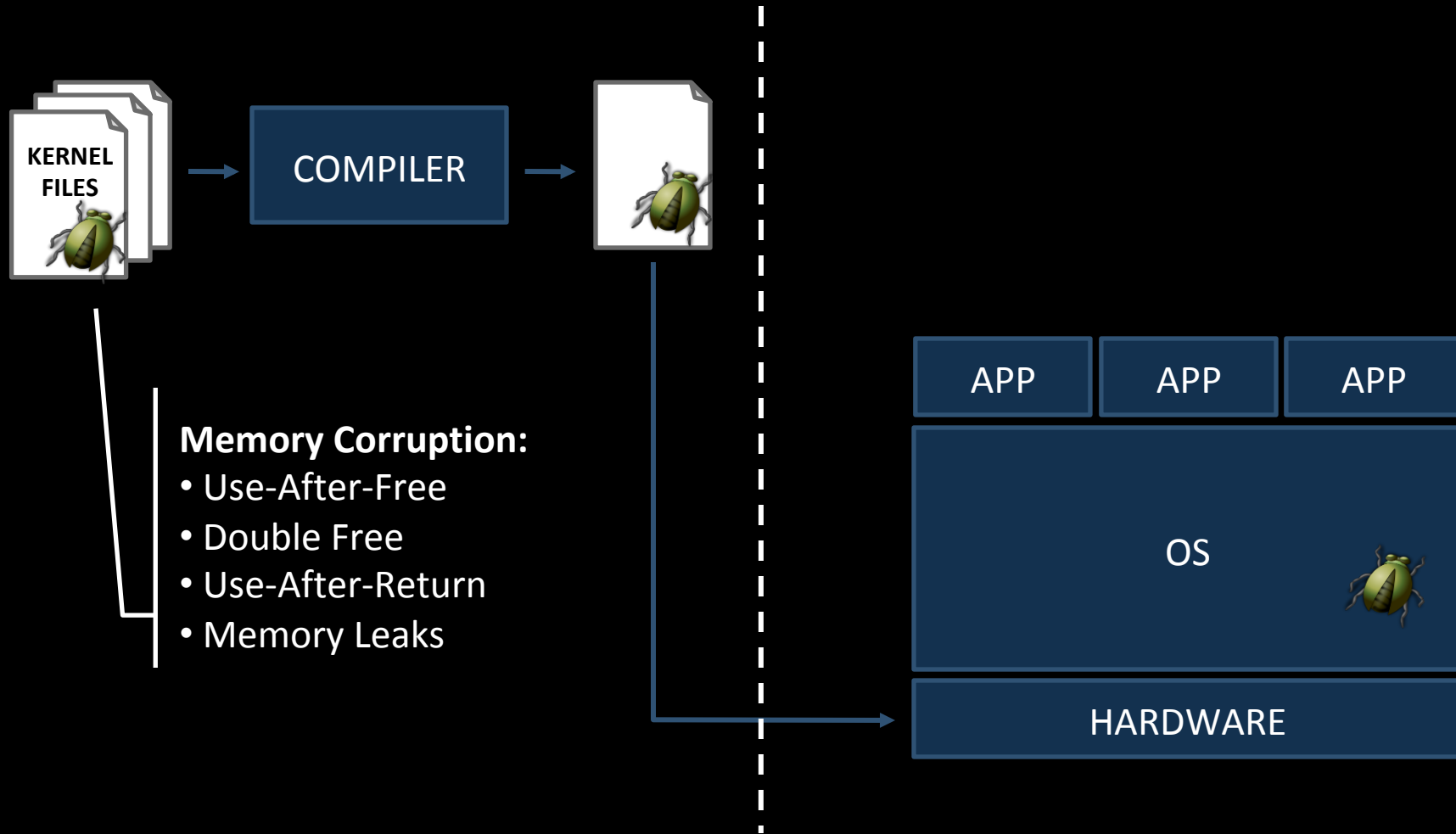
Cyber Security Center (CYSEC)
Technische Universität Darmstadt

Lucas Davi

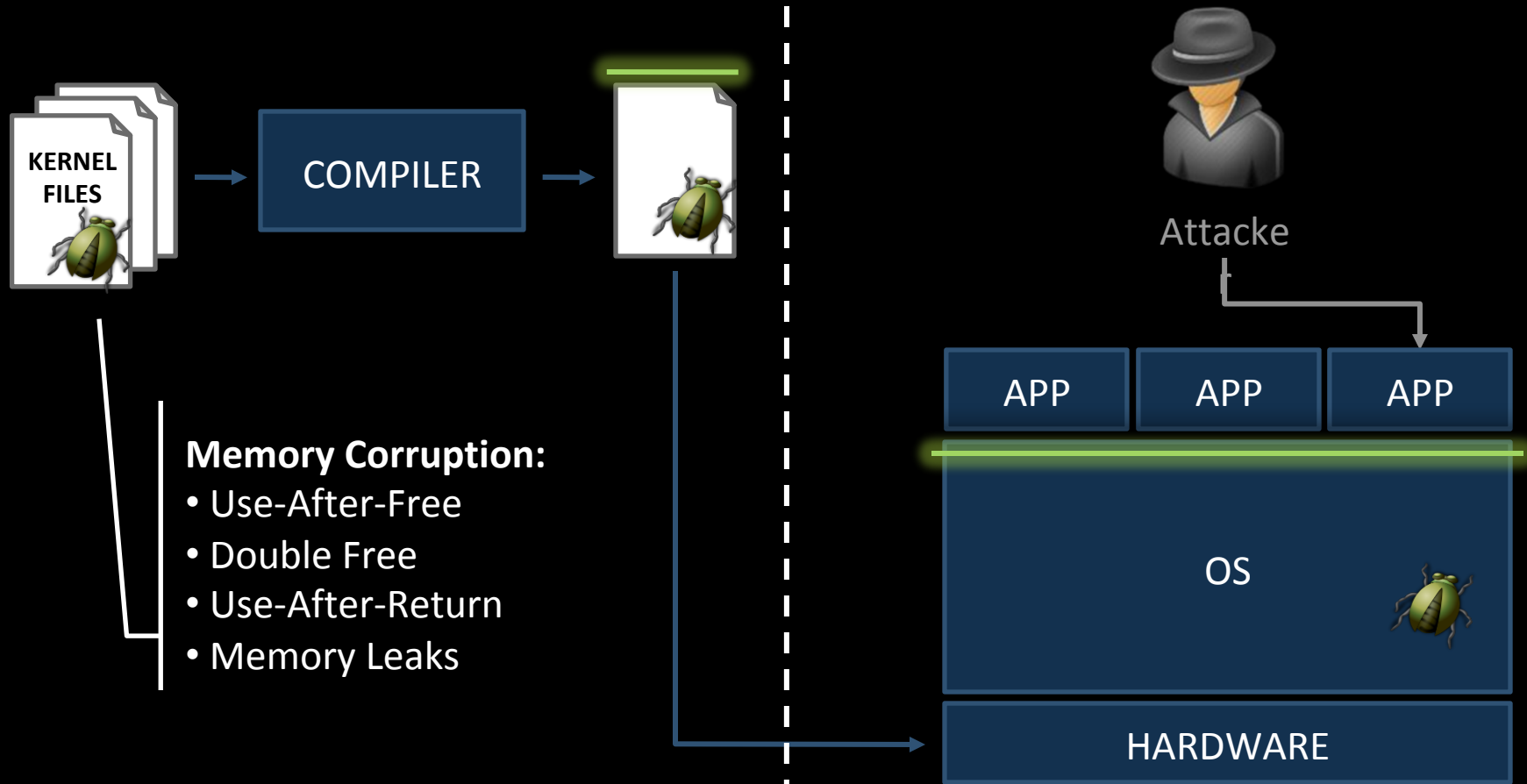
Universität of Duisburg-Essen

Why Static Analysis?

Big Picture



Big Picture



Memory Corruption:

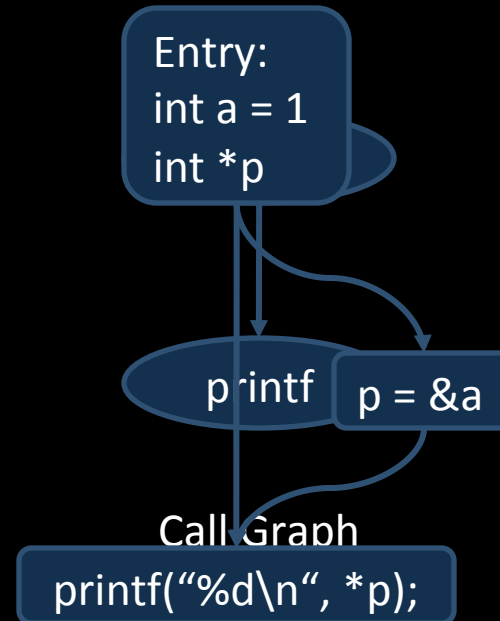
- Use-After-Free
- Double Free
- Use-After-Return
- Memory Leaks

Static Analysis at compile time.

Dynamic Analysis at run time.

Data-Flow Analysis: Graphs

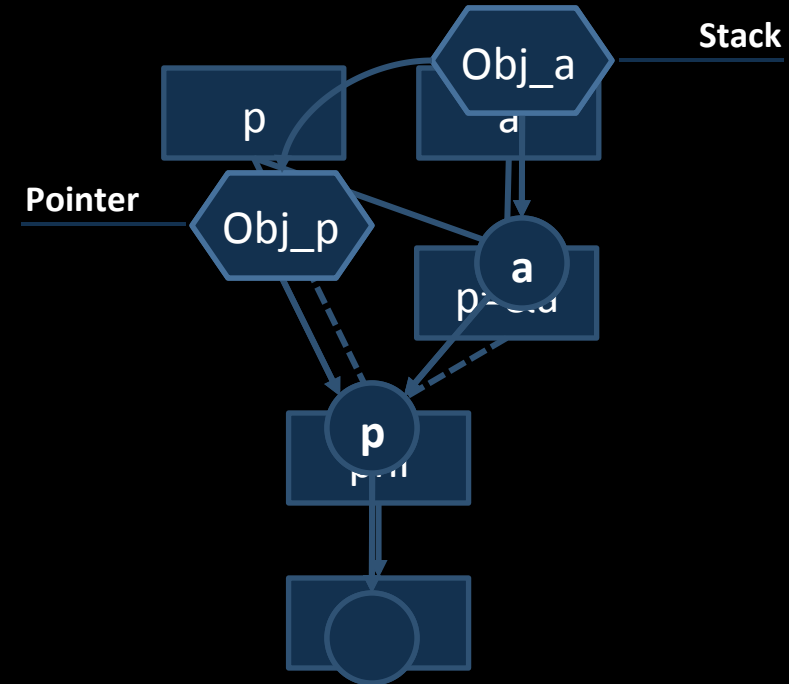
```
void main() {  
    int a = 1;  
    int *p;  
  
    if (a != 0)  
        p = &a;  
  
    printf("%d\n", *p);  
}
```



Control-Flow Graph


Data-Flow Analysis: Graphs

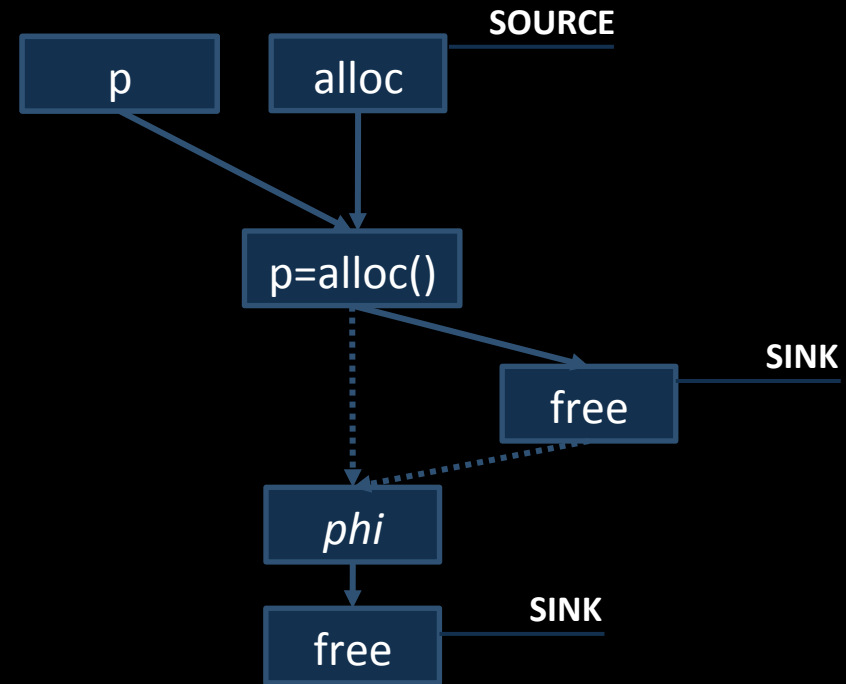
```
void main() {  
    int a = 1;  
    int *p;  
  
    if (a != 0)  
        p = &a;  
  
    printf("%d\n", *p);  
}
```



Control Flow Graph

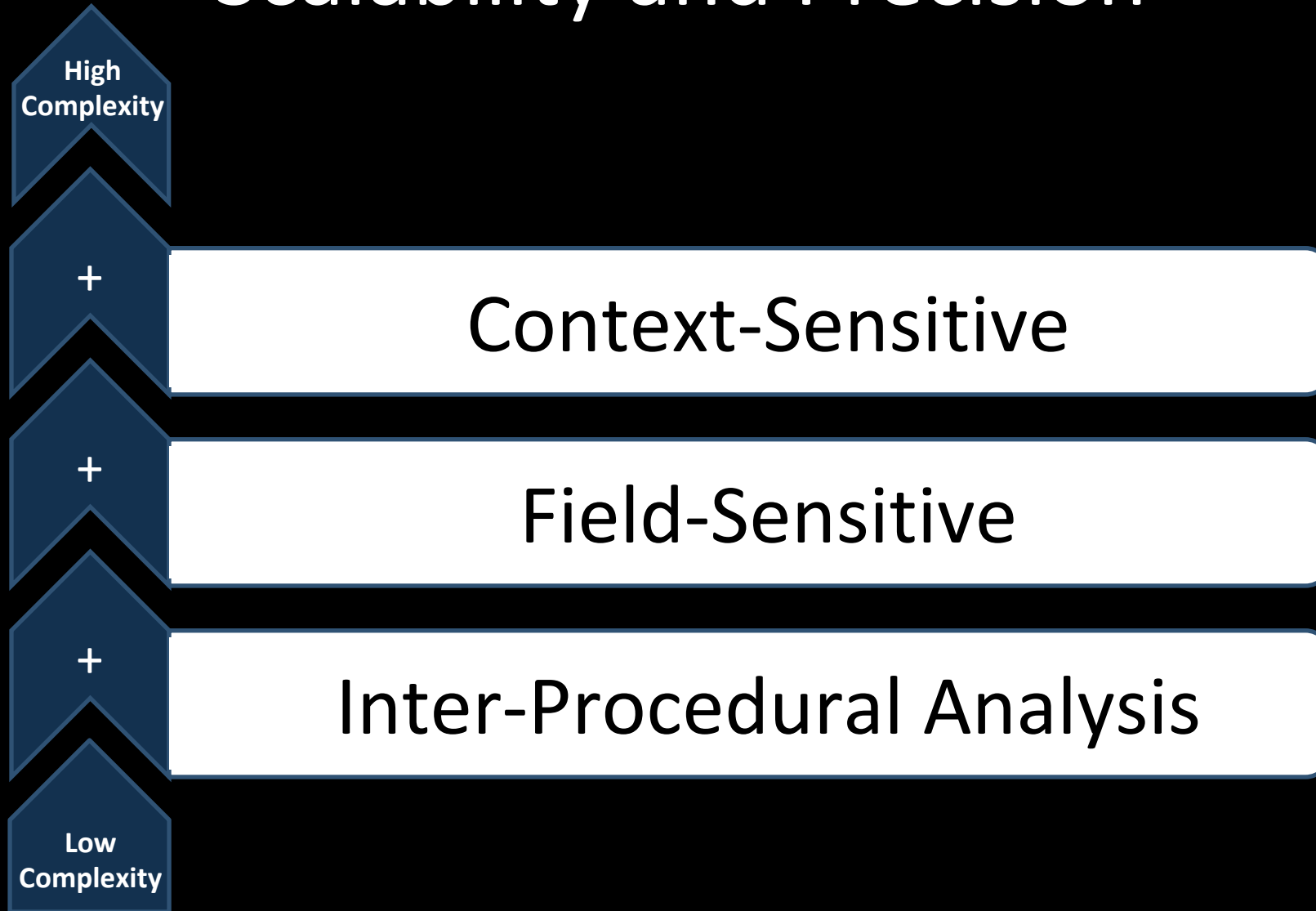
Source-Sink Analysis

```
void main() {  
    int *p = alloc();  
  
    if (p != NULL)  
        free(p);  
  
    free(p);   
}
```

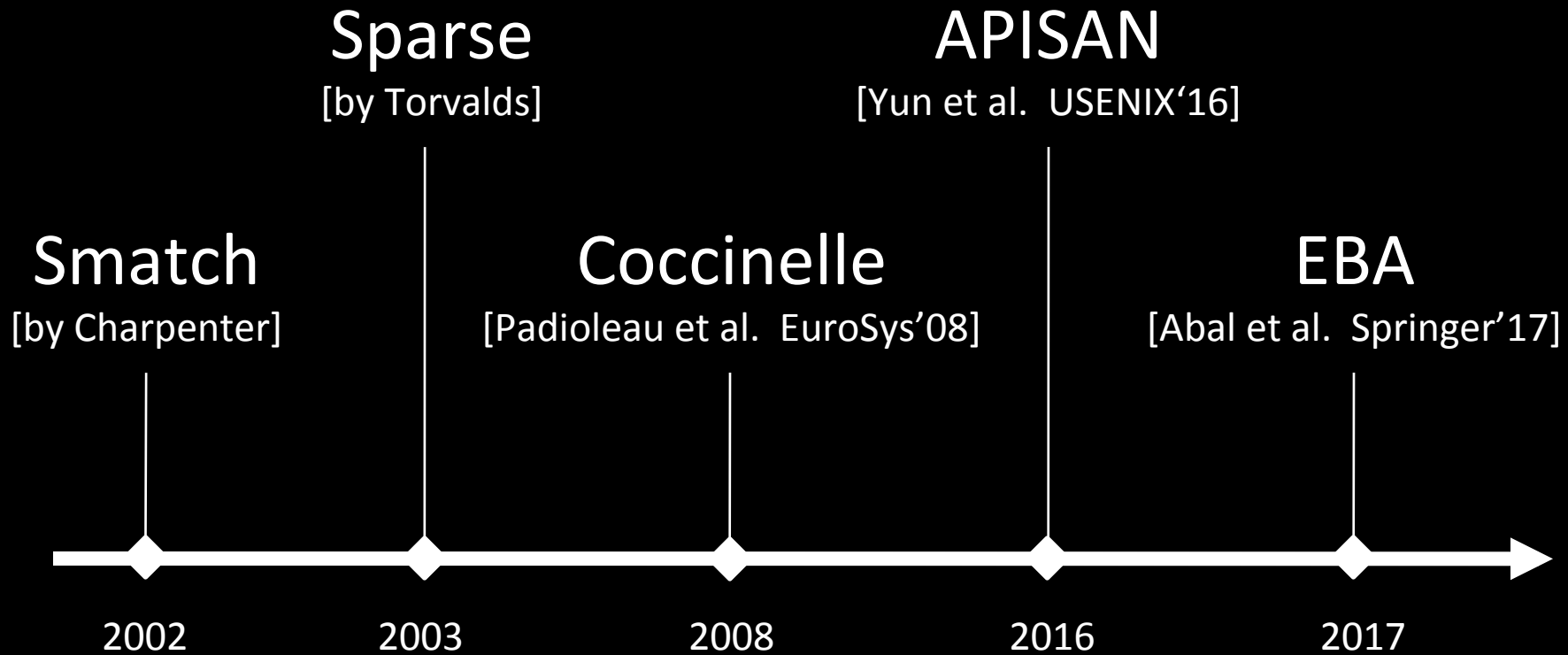


There exists a path that reaches two sinks.

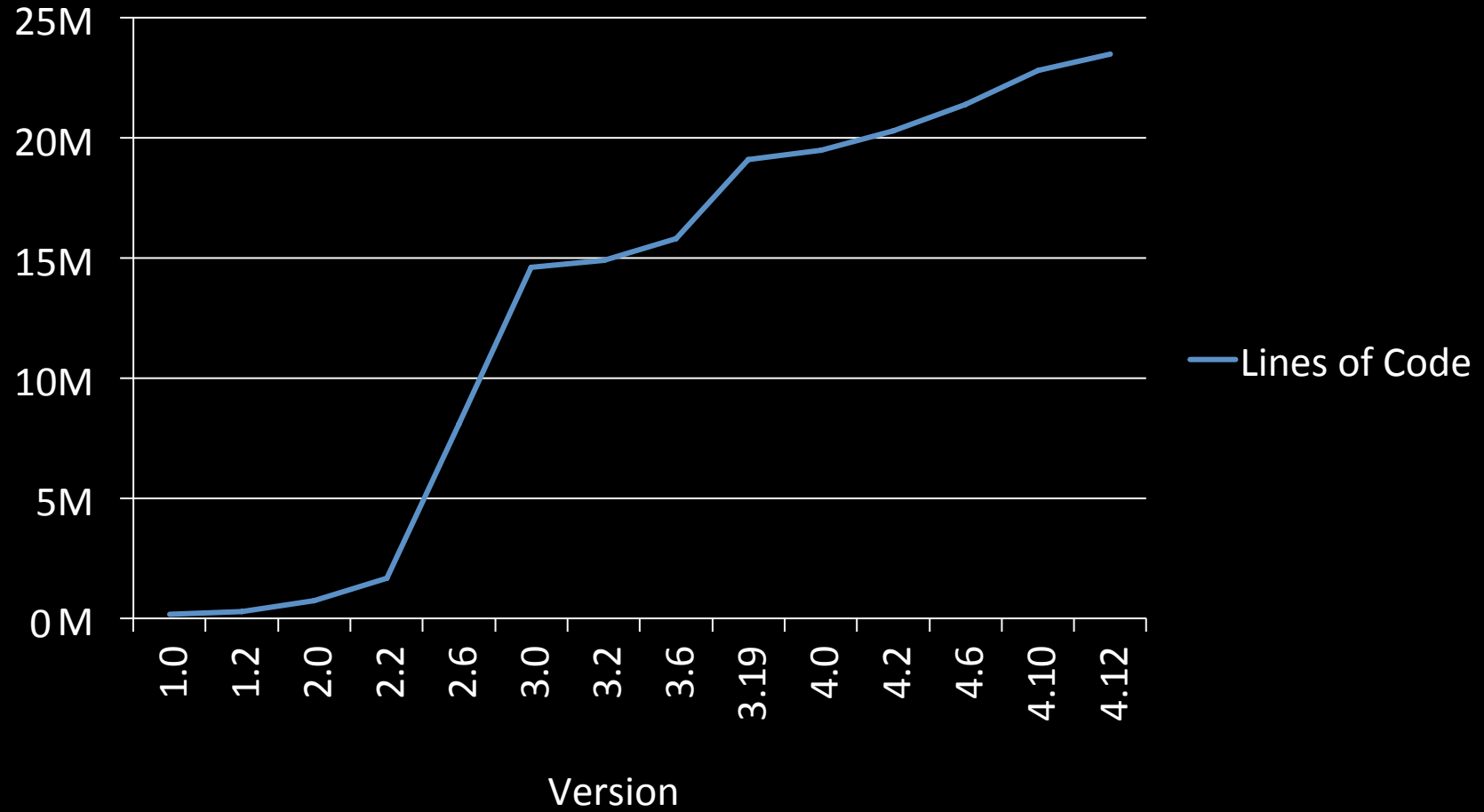
Scalability and Precision



Kernel Static Analysis - History



Linux Kernel: Lines of Code

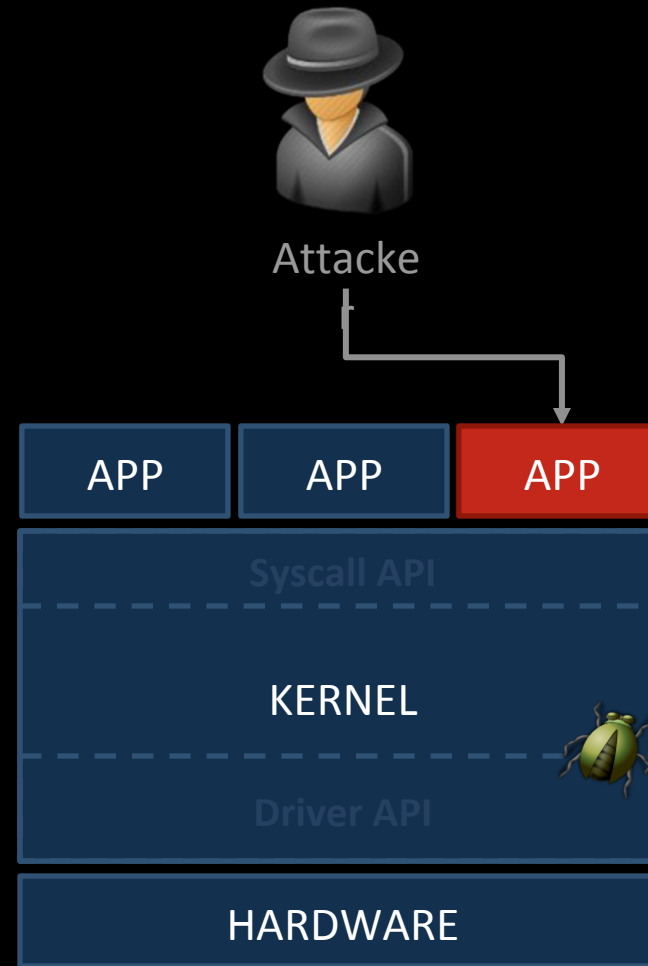


Contributions and Challenges

Analysis Format	Transformation of the kernel source into the LLVM intermediate representation
Large Codebase	Scaling inter-procedural Data-Flow Analysis to millions of kernel code lines
Expandability	A modular design, adding new checkers is straightforward
Data-Flow Analysis	Combine existing and novel approaches for inter-procedural bug checkers
Different Kernel Versions	Reports of different kernel versions can be managed using a web-based interface

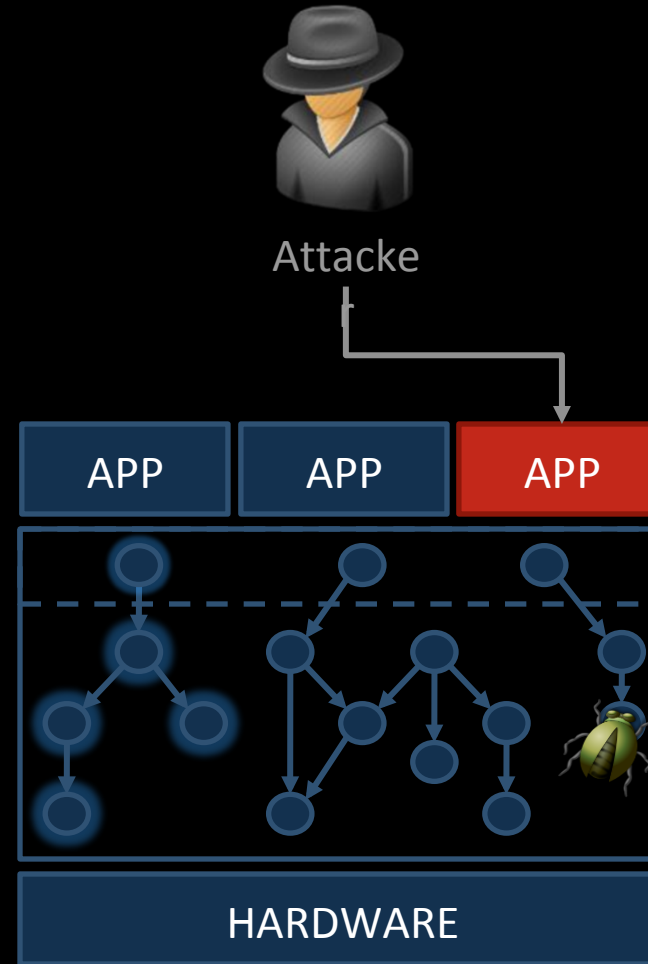
Enabling Data-Flow Analysis

- **Idea:** Partitioning the kernel along the system call API
- System calls are the interface between user- and kernel-space
- Vulnerabilities have to be exploited through system calls



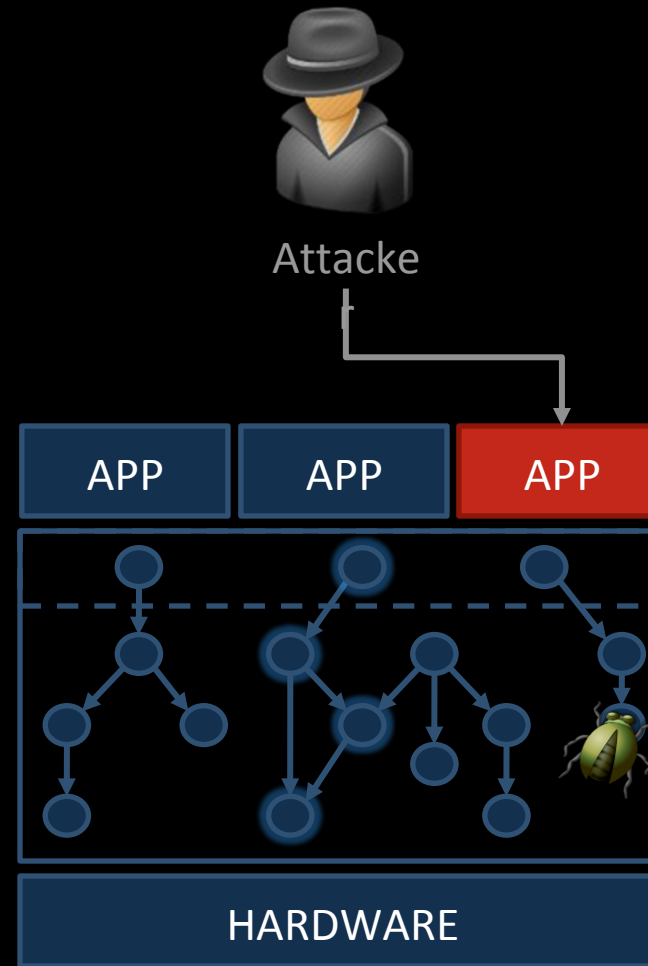
Enabling Data-Flow Analysis

- **Idea:** Partitioning the kernel along the system call API
- System calls are the interface between user- and kernel-space
- Vulnerabilities have to be exploited through system calls



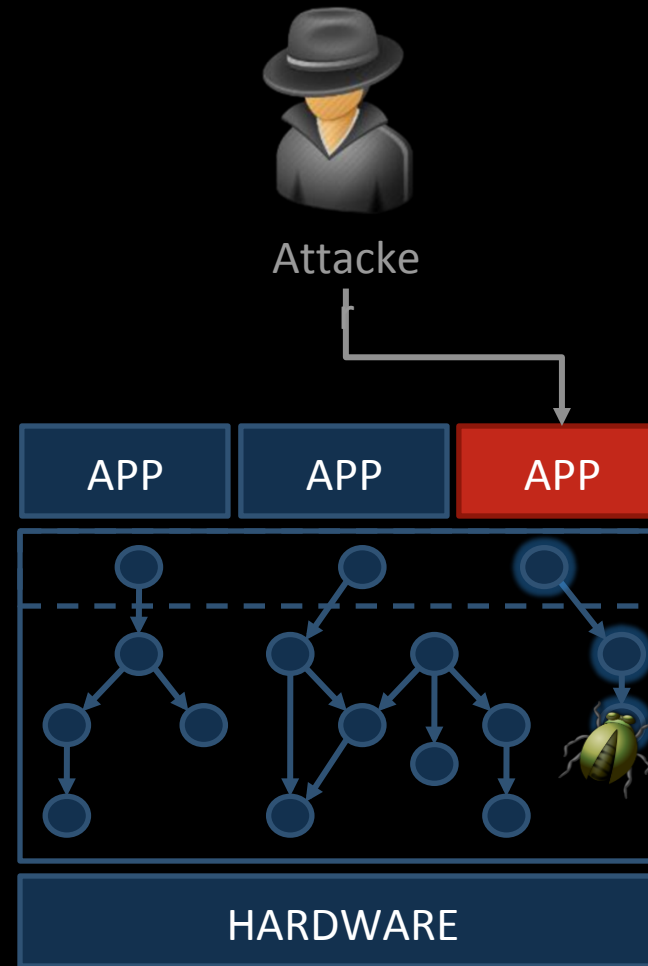
Enabling Data-Flow Analysis

- **Idea:** Partitioning the kernel along the system call API
- System calls are the interface between user- and kernel-space
- Vulnerabilities have to be exploited through system calls

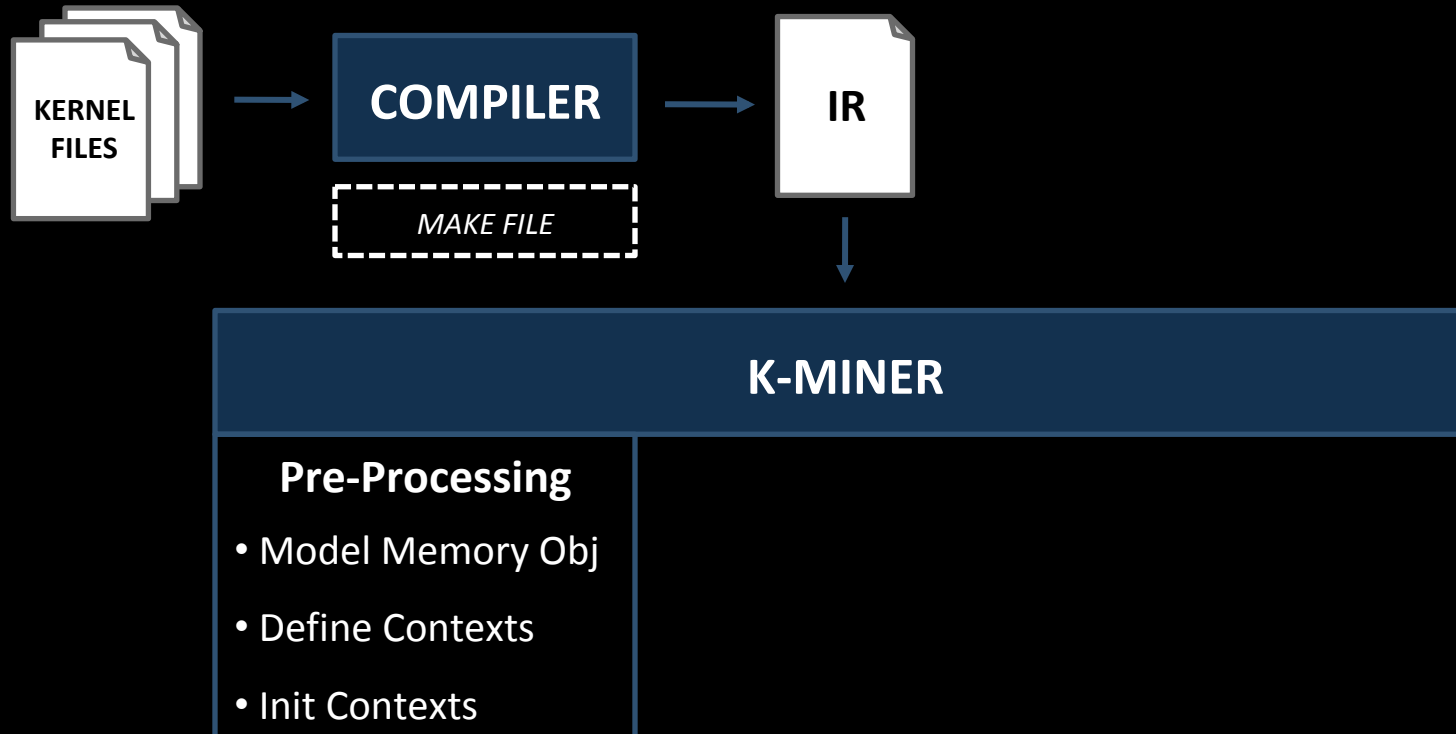


Enabling Data-Flow Analysis

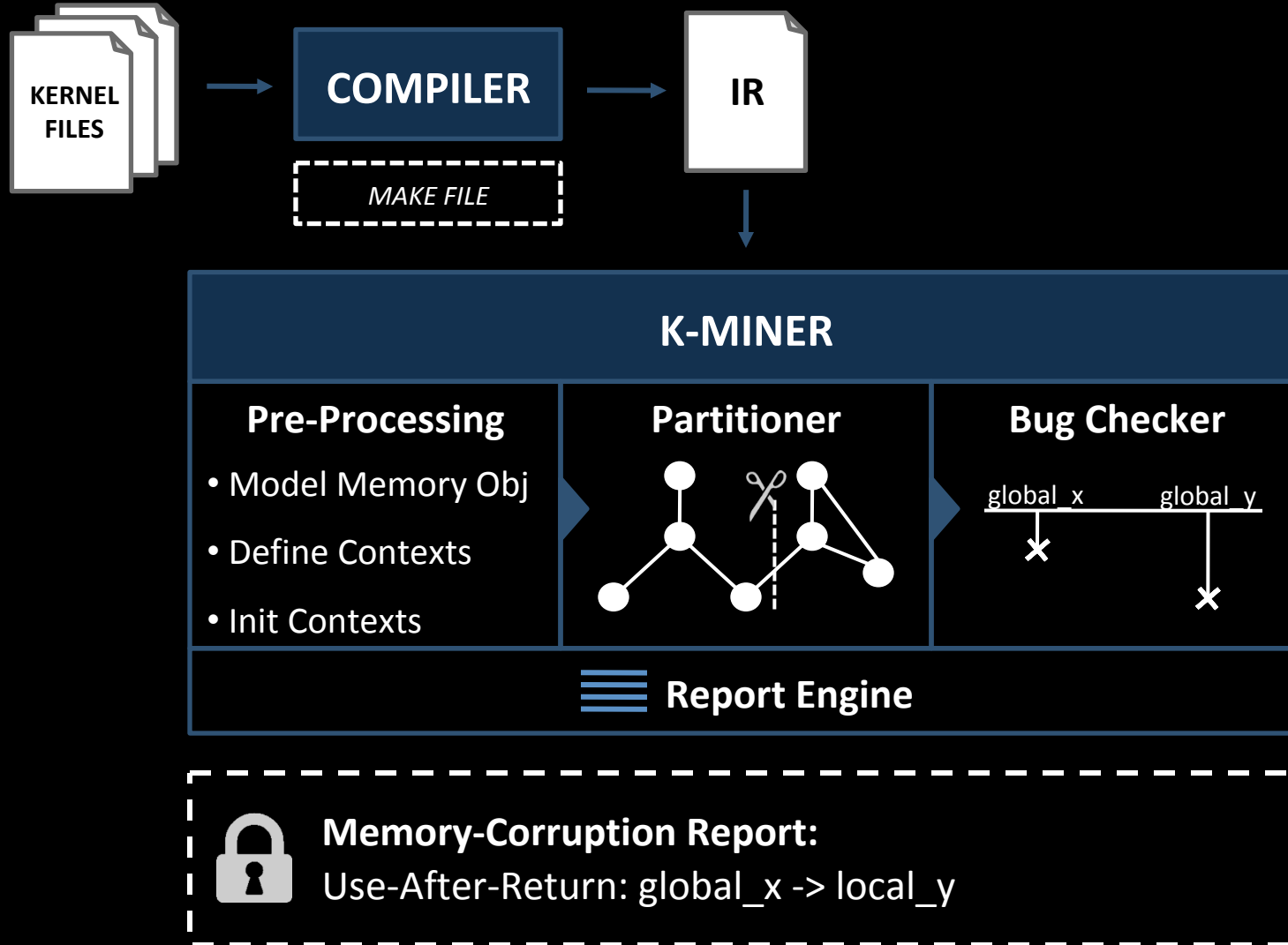
- **Idea:** Partitioning the kernel along the system call API
- System calls are the interface between user- and kernel-space
- Vulnerabilities have to be exploited through system calls



Design and Workflow



Design and Workflow



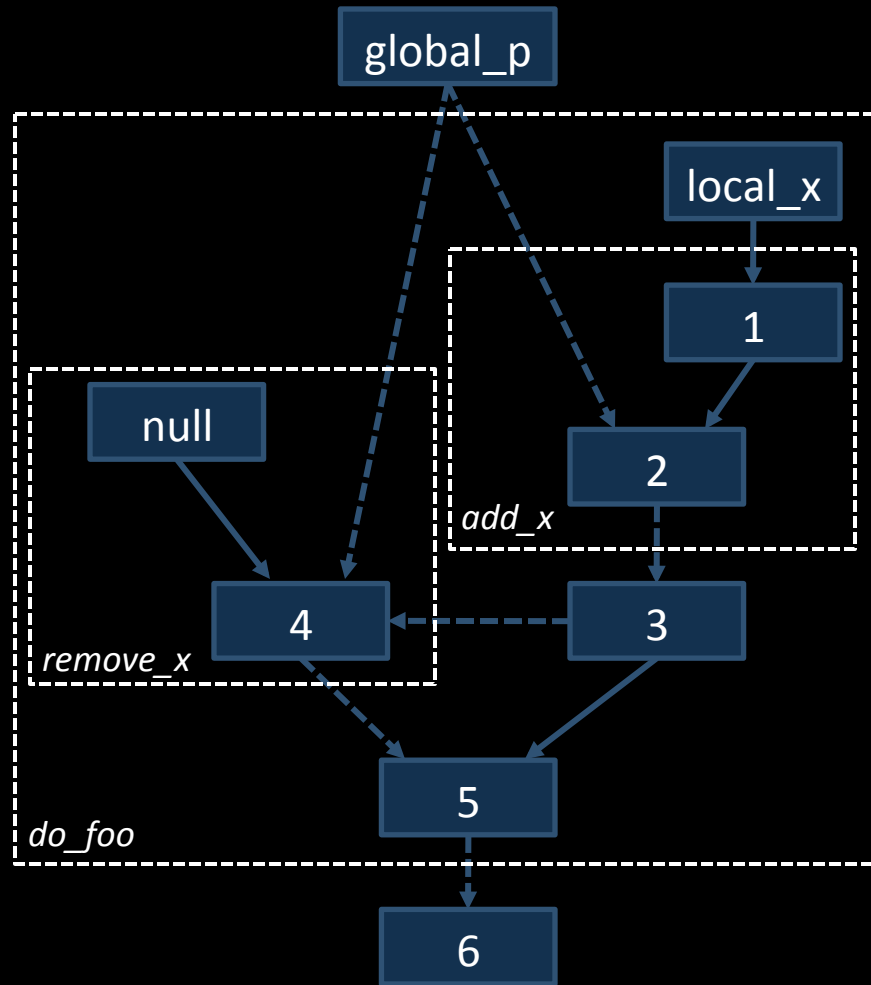
Implementation Details

- K-Miner builds on top of LLVM and SVF
- Environment Setup:
 - Register allocation sites by using a list of kernel allocation functions for dynamically allocated objects
- Context Handling:
 - Defines syscalls and initcalls contexts
 - Performs a call-graph analysis and pointer analysis
 - Multi-Level reduction of relevant partitions
- Bug Checker:
 - Exploit LLVM's Pass-Infrastructure
 - Covers Use-After-Return, Double-Free and Memory-Leaks
 - Performs particular analysis and several validation checks

Use-After-Return Checker

```
void sys_foo() {  
    do_foo()  
    return  
}  
  
void do_foo() {  
    int local_x = 1  
    add_x(&local_x)  
    if(cond())  
        remove_x()  
    return  
}  
  
void add_x(int *p) {  
    global_p = p  
}  
  
void remove_foo() {  
    global_p = NULL  
}
```

⑥
③
⑤
①
②
④

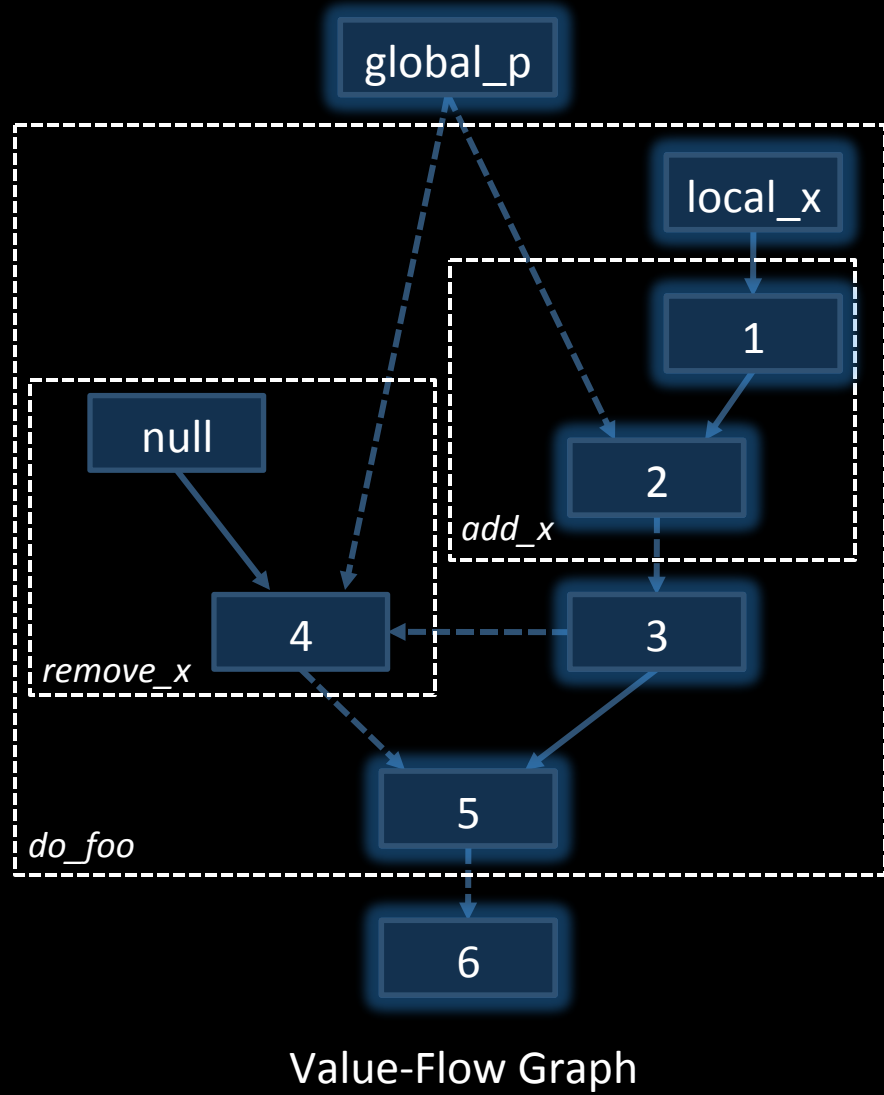


Value-Flow Graph

Use-After-Return Checker

```
void sys_foo() {  
    do_foo()  
    return  
}  
  
void do_foo() {  
    int local_x = 1  
    add_x(&local_x)  
    if(cond())  
        remove_x()  
    return  
}  
  
void add_x(int *p) {  
    global_p = p  
}  
  
void remove_foo() {  
    global_p = NULL  
}
```

⑥
③
⑤
①
②
④



Value-Flow Graph

Evaluation

- **Coverage:**
 - Four Kernel Versions
 - ca. 300 system calls
- **Scalability:**
 - \emptyset 25 min. per system call
 - \emptyset 12 GB memory usage
- **Real-world Impact:**
 - CVE-2014-3153 (UAR)
 - CVE-2015-8962 (DFree)
- **Extensibility:**
 - Additional bug checkers can be added as Passes

Version	MLC Functions	Run-Time Global	Memory	Locks	MLC Total	DFree
v3.19	1557/99396	6.9424/837623	GB 69677/410554	3/17148/5680390		
v4.02	2086/1041435	5.62522/14773	GB 93091/436451	2/107894/501396		
v4.06	2208/1041502	5.54526/14979	GB 99473/50052	2/130413/61073471		
v4.10	2246/1207312	4.14557/10580	GB 98730/518394	1/128253/7091277		

Kernel Report for sys_futex

v4.2

Checker Results

Use-After-Return Checker Results

6 Leaks found

took 508 sec

937 Variables analyzed



Use-After-Free Checker Results

7 Leaks found

took 19 sec



Double-Free Checker Results

0 Leaks found

took 67 sec



Minimization

Kernel Reduction

	Actual	Relevant	Original
Functions	2111	2109	104248
Global Variables	196	196	72712

Show Graph

2. Memory Leak:

	Dangling Pointer	Local Variable
Variable Name	call3	q
Function	futex_init	futex_wait_requeue_pi
File	kernel/futex.c	kernel/futex.c
Line	3036	2554

Systemcall to Local Variable

```
→ Sys_futex  
→ SYSC_futex  
→ do_futex  
→ futex_wait_requeue_pi
```

DanglingPointer to OutOfScope

```
futex_init  
↗ assignment (ln: 3036)  
← futex_init  
→ SYSC_futex  
→ do_futex  
 . → futex_wait_requeue_pi  
 . . → futex_wait_setup  
 . . . → queue_lock  
 . . . . → hash_futex  
 . . . . . = hash_futex  
 . . . . . ↗ assignment (ln: 1747)  
 . . . . . = queue_lock  
 . . . . ↗ assignment (ln: 2135)
```

LocalVariable to OutOfScope

```
futex_wait_requeue_pi  
= futex_wait_queue_me  
 . = queue_me  
 . . = plist_add  
 . . . = list_add_tail  
 . . . . = _list_add  
 . . . . . ↗ assignment (ln: 45)  
 . . . . . ← _list_add  
 . . . . ← list_add_tail  
 . . ← plist_add  
 . ← queue_me  
← futex_wait_queue_me  
→ handle_early_requeue_pi_wakeup
```

Kernel Report

Memory Leaks

Type	Function	File	Line	Checked
UseAfterFree	sget	fs/super.c	465	
UseAfterFree	expand_fdtbl	fs/file.c	149	
UseAfterFree	kmemdup	mm/util.c	113	
UseAfterFree	hib_submit_io	kernel/power/swap.c	264	
UseAfterFree	kstrndup	mm/util.c	93	✓
UseAfterFree	create_worker	kernel/workqueue.c	1698	
UseAfterFree	init_workqueues	kernel/workqueue.c	5279	
UseAfterFree	load_image_lzo	kernel/power/swap.c	1172	
UseAfterFree	store_rps_dev_flow_table_cnt	net/core/net-sysfs.c	787	
UseAfterFree	check_partition	block/partitions/check.c	147	
UseAfterFree	init_workqueues	kernel/workqueue.c	5280	
UseAfterFree	init_workqueues	kernel/workqueue.c	5278	
UseAfterFree	init_workqueues	kernel/workqueue.c	5282	

<https://github.com/ssl-tud/k-miner>

<https://github.com/ssl-tud/k-miner>

Questions?