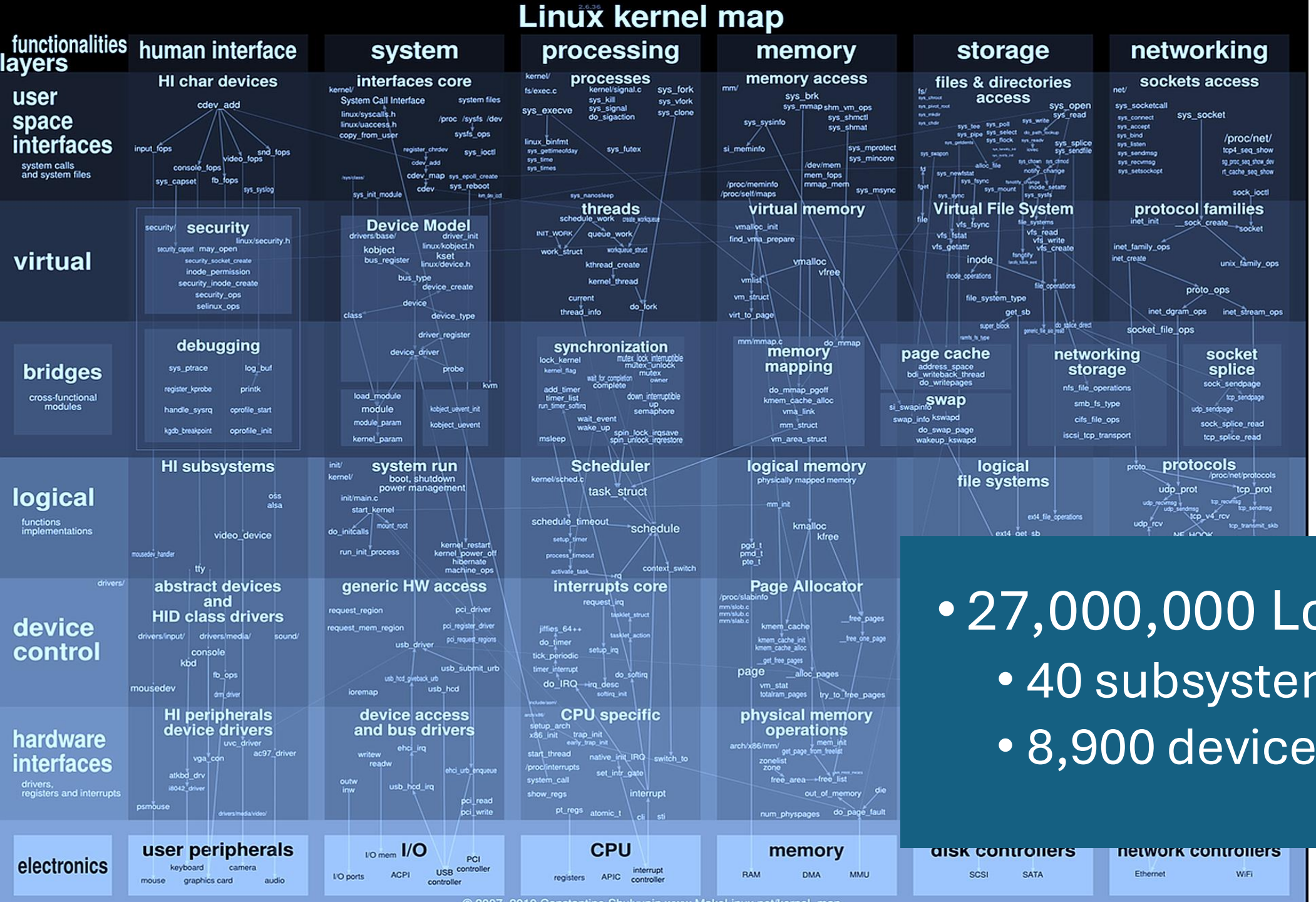
CS6465: Advanced Operating System Implementation

Lecture 11 – Bringing Isolation to the Kernel

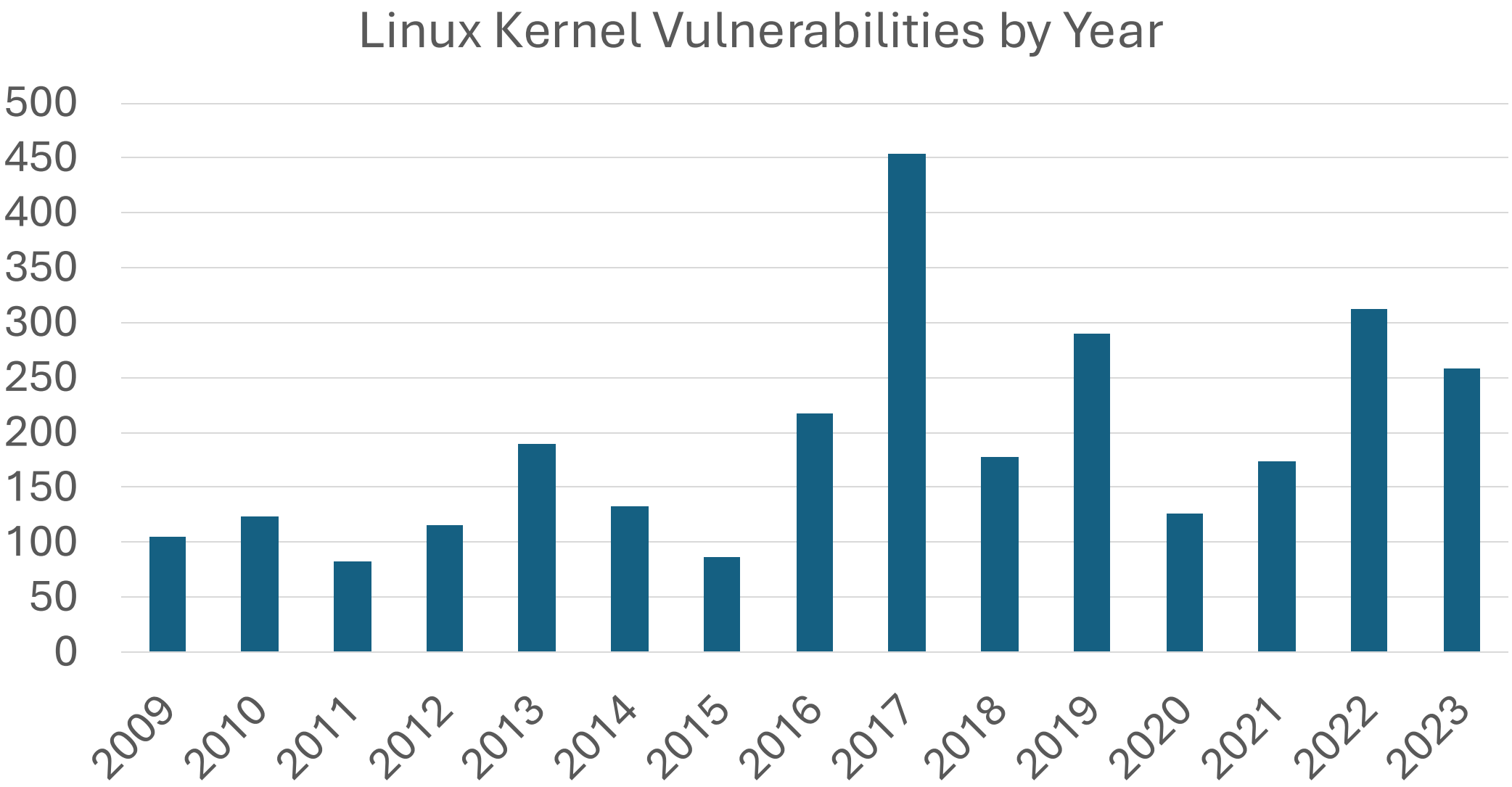
Anton Burtsev

November 2024



- 27,000,000 LoC
  - 40 subsystems
  - 8,900 device drivers

Problem: Security


Linux Kernel Vulnerabilities by Year
500
450
400
350
300
250
200
150
100
50
0
2009
2010
2011
2012
2013
2014
2015
2016
2017
2018
2019
2020
2021
2022
2023

Example

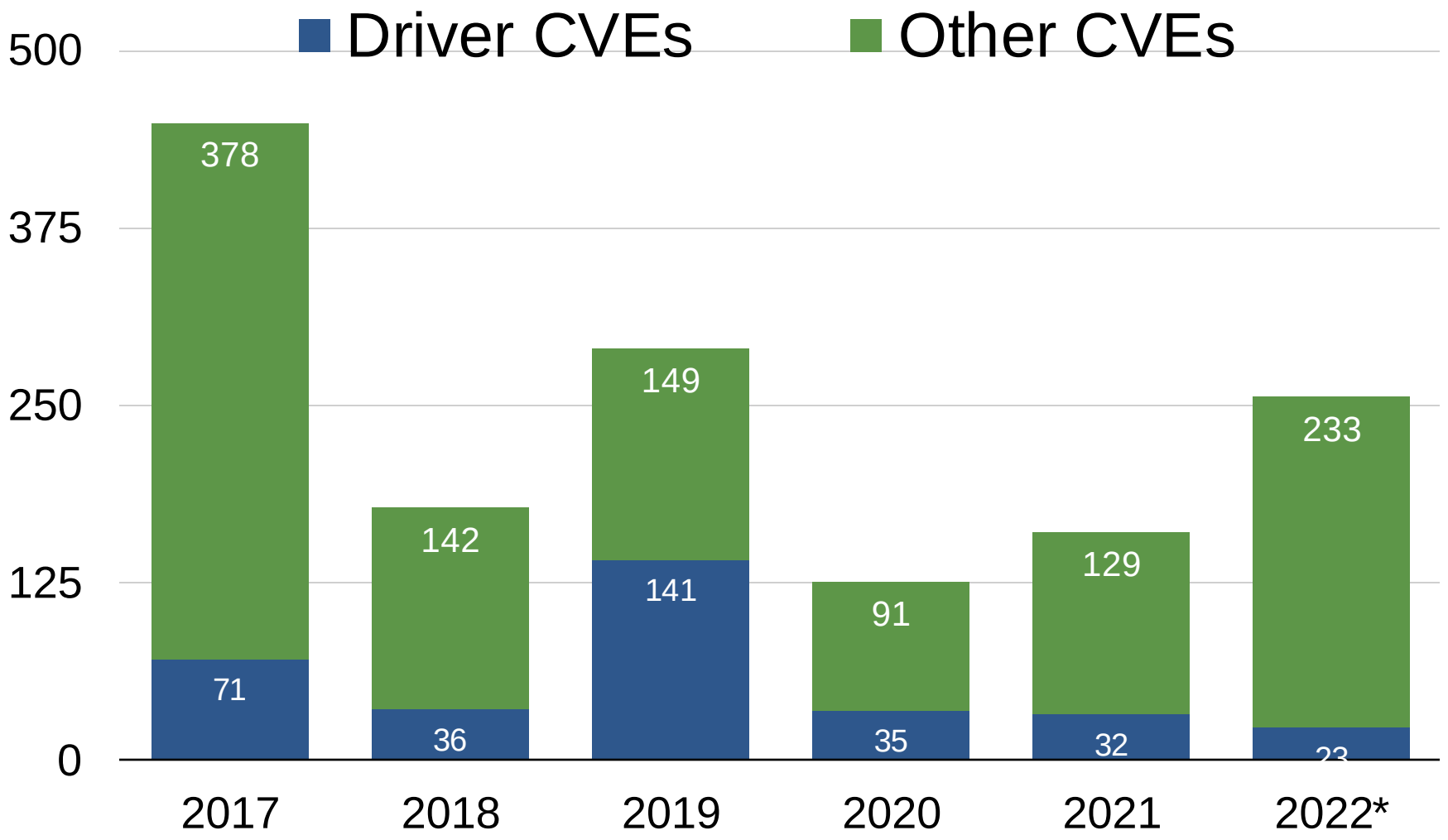
```
static bool dccp_new (...) {
  struct dccp_header _dh, *dh;

- skb_header_pointer(skb, dataoff, sizeof(_dh), &dh);
+ skb_header_pointer(skb, dataoff, sizeof(_dh), &_dh);
};
```

Stack smash

Correct

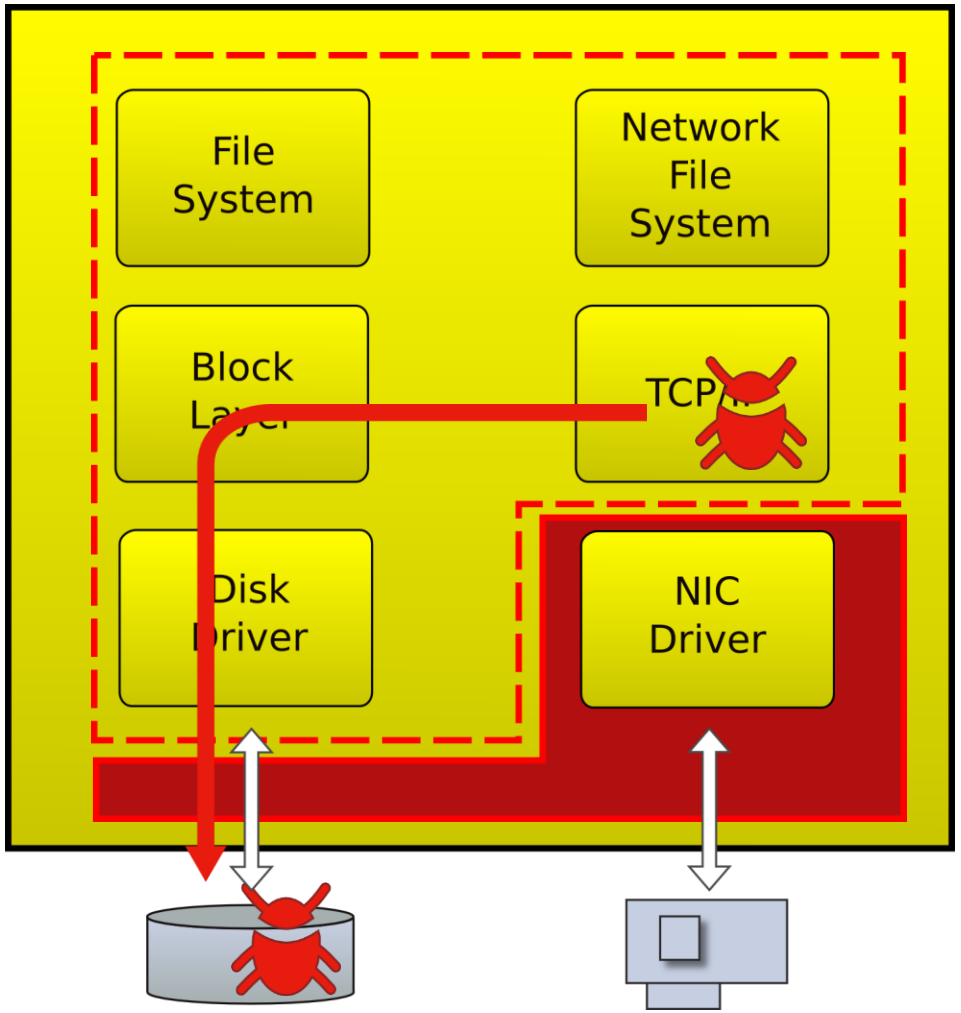
- Remote exploit in Linux network firewall
  - Arbitrary code execution
  - Linux Kernel v 3.0 (June, 2011) – 3.13.6 (March, 2014)
  - CVE-2014-2523

Large fraction of CVEs are in drivers



Anatomy of a kernel exploit

- Exploit
- Persistence
- Rootkit

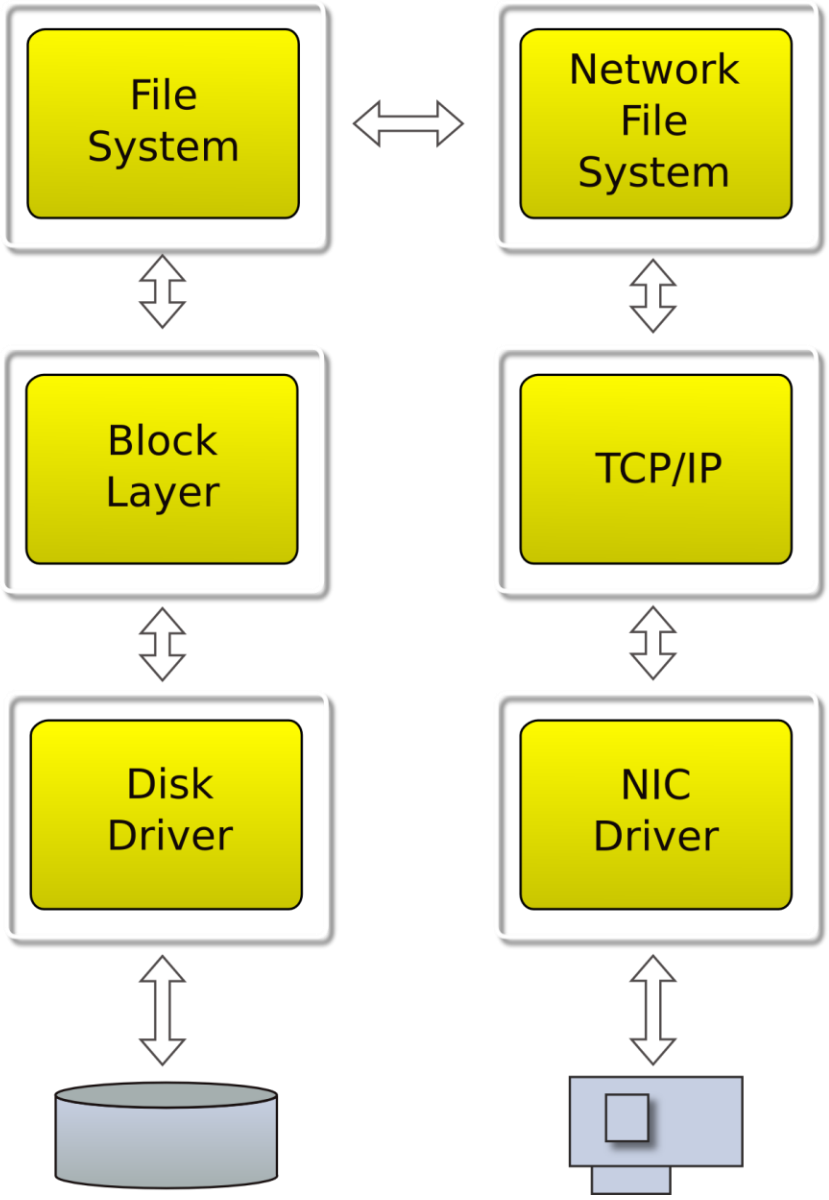


In a modern system, an attacker is **one kernel vulnerability away** from gaining complete control of the entire machine

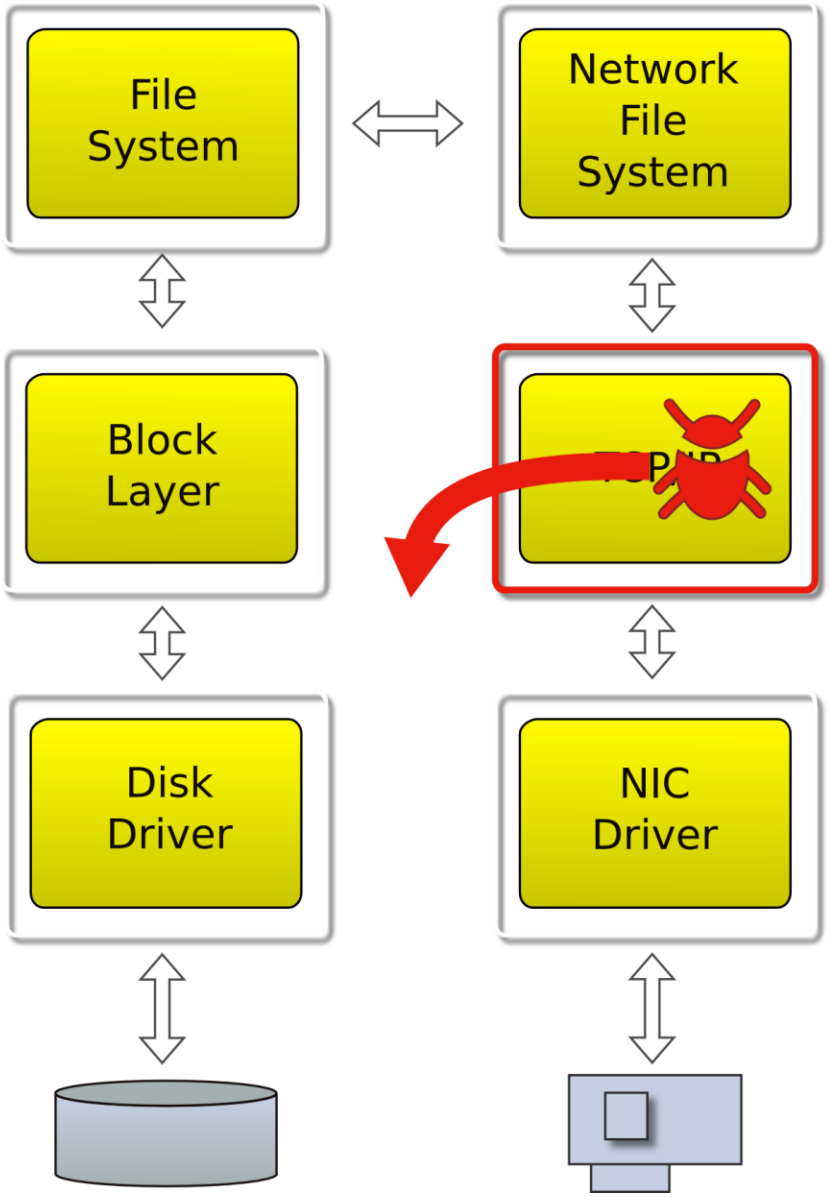
- Not going to change

Can we make our systems secure?

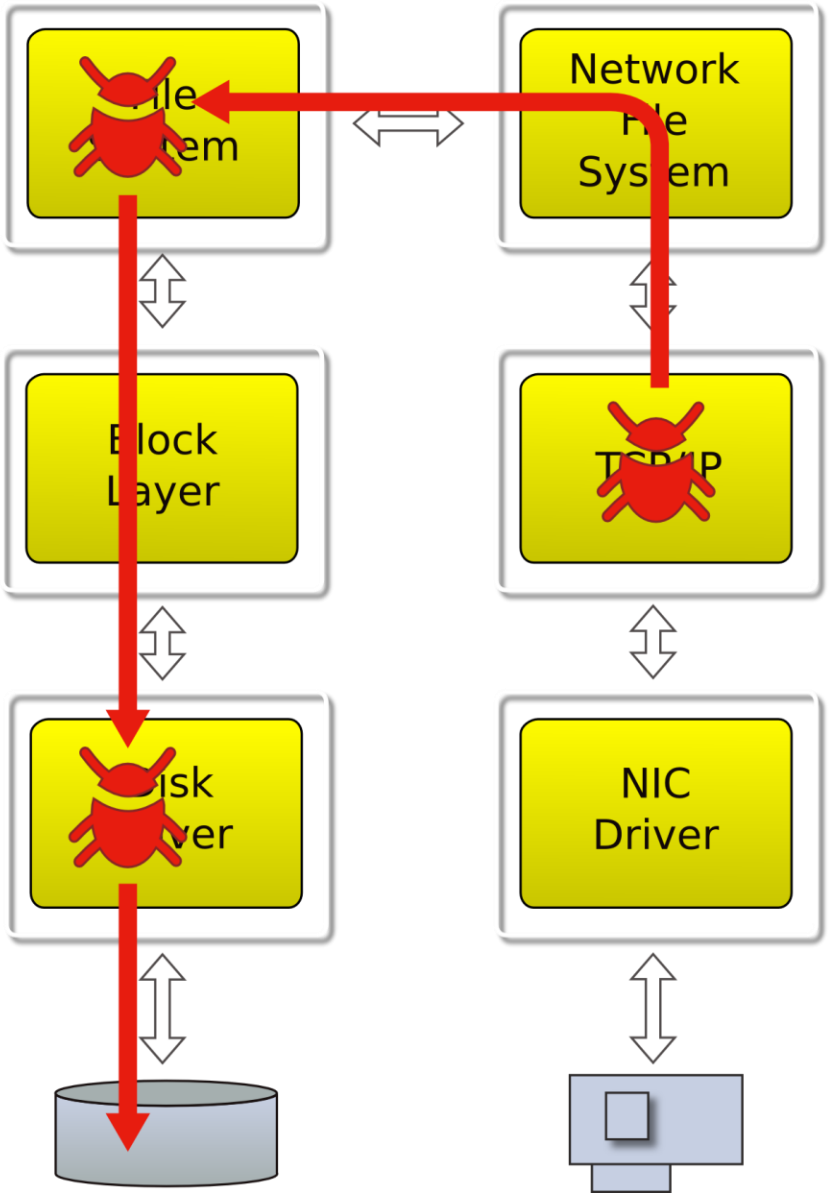
Isolation


File System
Network File System
Block Layer
TCP/IP
Disk Driver
NIC Driver

Isolation


File System
Network File System
Block Layer
Disk Driver
NIC Driver

Isolation



Isolation is effective

- Web browsers
- Mashup web pages
- Application containers

- Example: Google Chrome exploit
  - 9 steps
  - 6 vulnerabilities

Isolation is hard

- Commodity kernels are not built for isolation
  - Shared memory
  - Call/return programming model
  - Complex interfaces

- Microkernels tried and failed
  - Massive engineering effort
  - Performance issues

Our goal: decomposed kernel

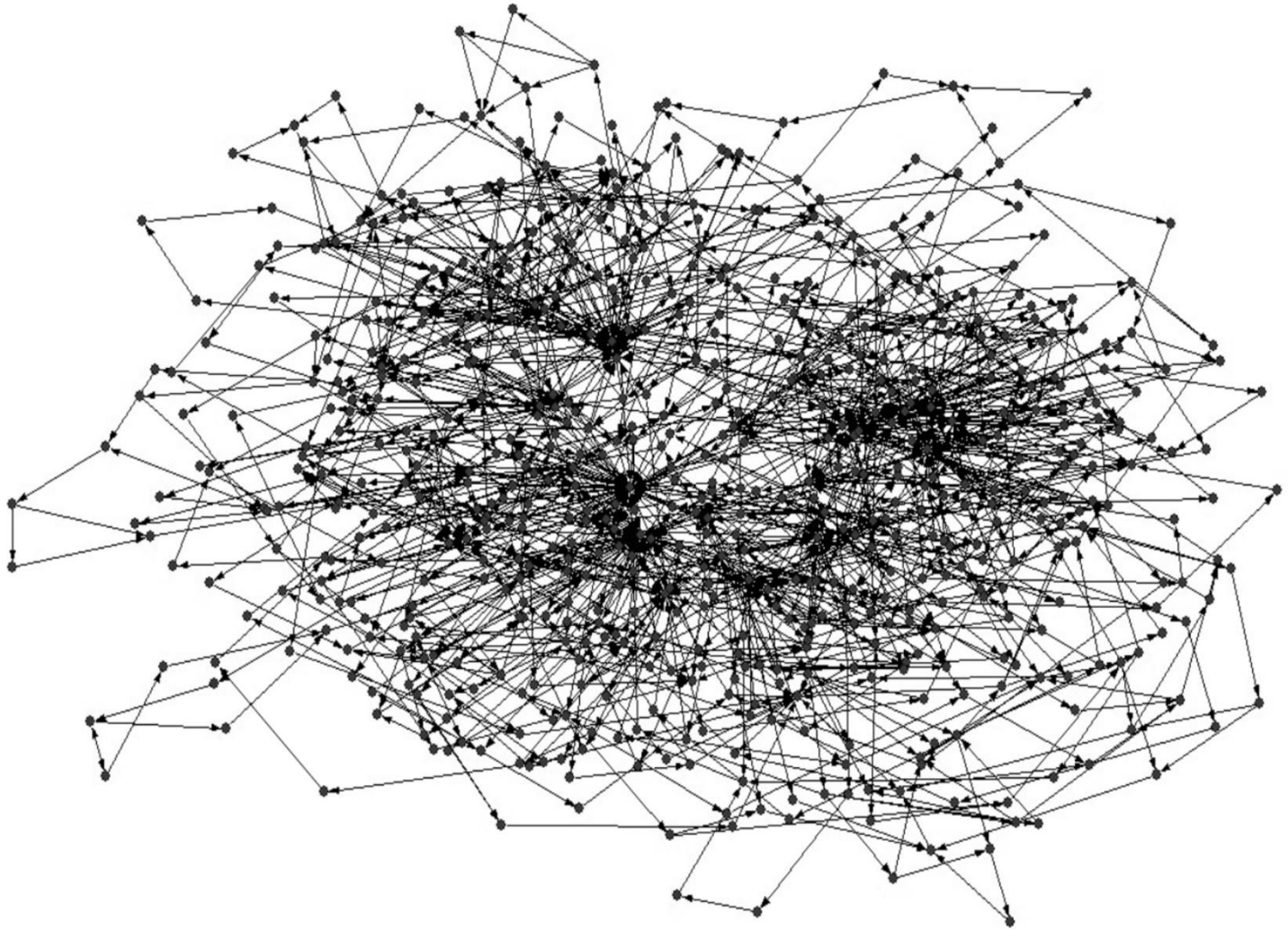
- Strongly isolated environment
- Explicit access control for each resource
- Reuse of unmodified code
- Fast

Key Problems

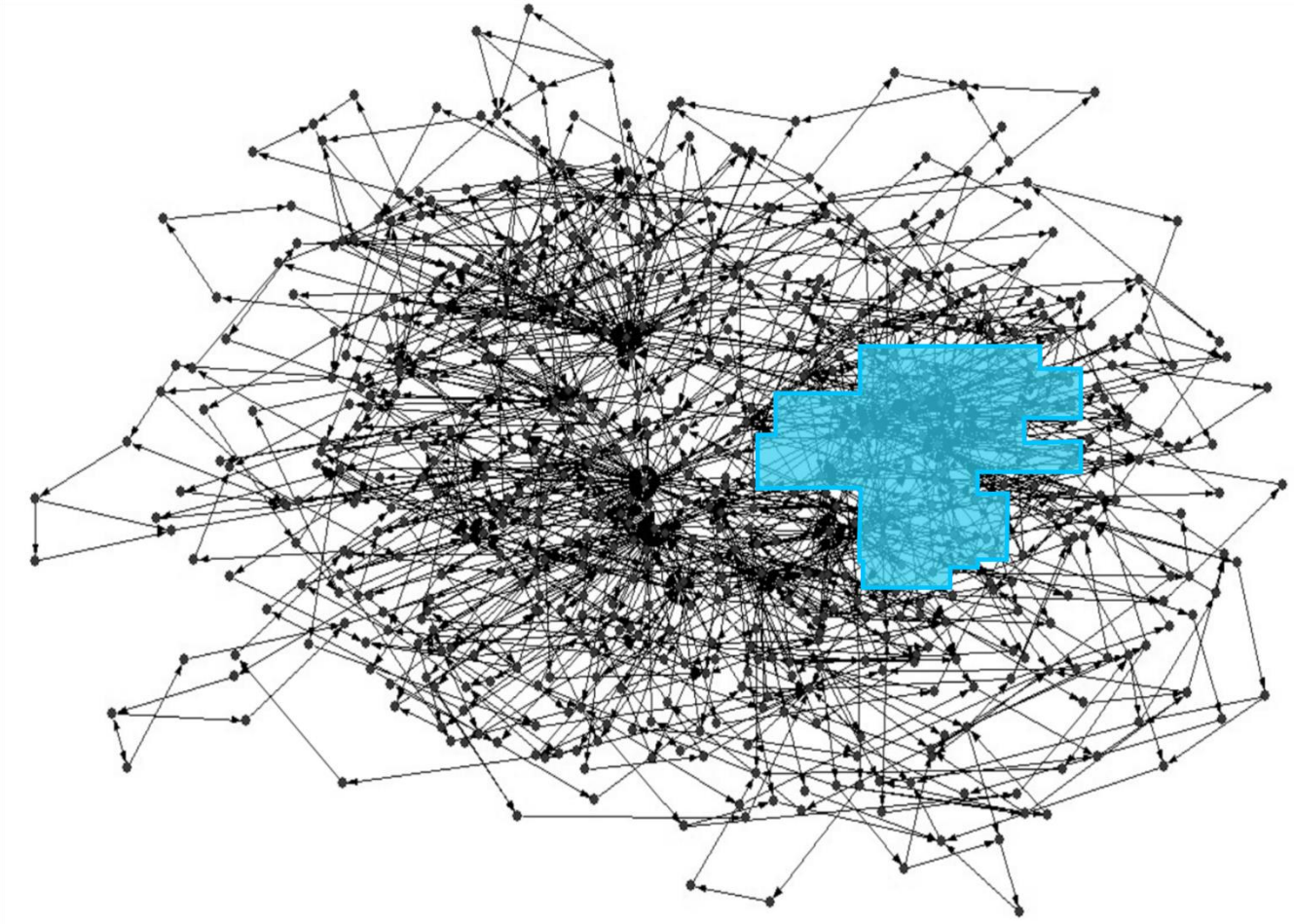
- Performance
- Semantic complexity of isolating subsystems

Performance

- Historically context switches are expensive:
  - 814 cycles for a cross-subsystem invocation on Intel CPUs1 It takes roughly 2000 cycles to send a network packet on Linux
  - 100 cycles on DPDK
- Recent hardware and software primitives:
  - Intel Memory Protection Keys (MPKs)
  - ARM Memory Tagging Extensions (MTE)
  - ARM Pointer Authentication (PAC)
  - CHERI Morello
  - Rust



- Strong component part of the call graph of the Linux kernel2
- Roughly 0.1% of 453K nodes in 3.7.10 kernel



- Strong component part of the call graph of the Linux kernel2
- Roughly 0.1% of 453K nodes in 3.7.10 kernel

FINAL REPORT OF THE MULTICS KERNEL DESIGN PROJECT

by

M.D. Schroeder*
D.D. Clark
J.H. Saltzer
D.H. Wells

Software fault isolation with API integrity and multi-principal modules

Yandong Mao, Haogang Chen, Dong Zhou†, Xi Wang, Nickolai Zeldovich, and M. Frans Kaashoek
MIT CSAIL, †Tsinghua University IIIS

Unmodified Device Driver Reuse and Improved System Dependability via Virtual Machines

Joshua LeVasseur Volkmar Uhlig Jan Stoess Stefan Götz

J-Kernel: a Capability-Based Operating System for Java

Dingo: Taming Device Drivers

From L3 to seL4 What Have We Learnt in 20 Years of L4 Microkernels?

Kevin Elphinstone and Gernot Heiser

VirtuOS: an operating system with kernel virtualization

Ruslan Nikolaev, Godmar Back

Singularity: Rethinking the Software Stack

Galen C. Hunt and James R. Larus

Decaf: Moving Device Drivers to a Modern Language

Matthew J. Renzelmann and Michael M. Swift

Nooks: An Architecture for Reliable Device Drivers *

Michael M. Swift, Steven Martin, Henry M. Levy, and Susan J. Eggers

Tolerating Malicious Device Drivers in Linux

Silas Boyd-Wickizer and Nickolai Zeldovich
MIT CSAIL

Microdrivers: A New Architecture for Device Drivers

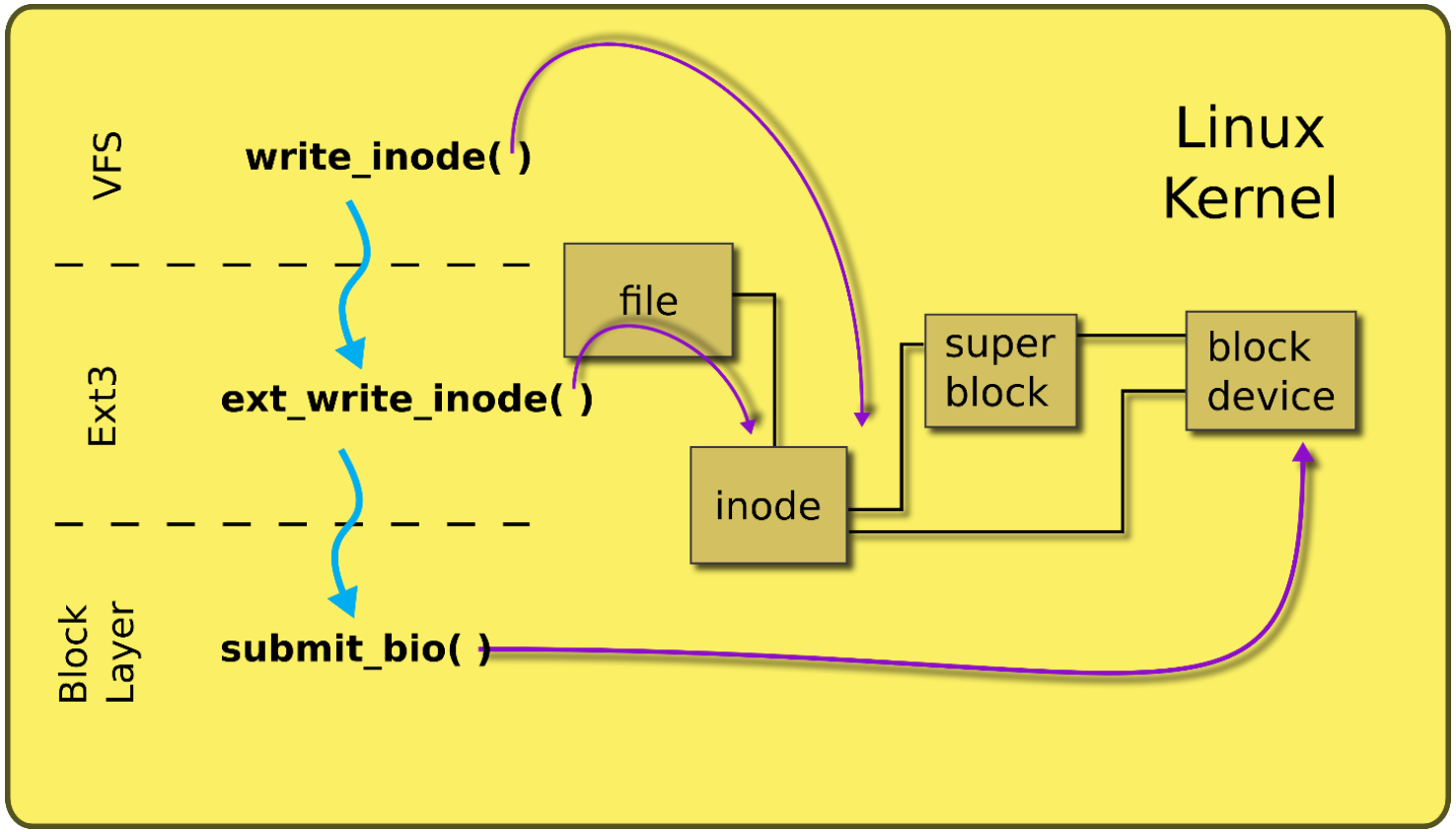
Vinod Ganapathy, Arini Balakrishnan, Michael M. Swift and Somesh Jha

MINIX 3: A Highly Reliable, Self-Repairing Operating System

Fine-Grained Fault Tolerance using Device Checkpoints

Asim Kadav, Matthew J. Renzelmann, Michael M. Swift

Commodity OS: Shared Object Space



LXFI: Software fault isolation

Software fault isolation with API integrity and multi-principal modules

Yandong Mao, Haogang Chen, Dong Zhou[†], Xi Wang, Nickolai Zeldovich, and M. Frans Kaashoek
MIT CSAIL, [†]Tsinghua University IIIS

ABSTRACT

The security of many applications relies on the kernel being secure, but history suggests that kernel vulnerabilities are routinely discovered and exploited. In particular, exploitable vulnerabilities in kernel modules are common. This paper proposes LXFI, a system which isolates kernel modules from the core kernel so that vulnerabilities in kernel modules cannot lead to a privilege escalation attack. To safely give kernel modules access to complex kernel APIs, LXFI introduces the notion of *API integrity*, which captures the set of contracts assumed by an interface. To partition the privileges within a shared module, LXFI introduces *module principals*. Programmers specify principals and API integrity rules through capabilities and annotations. Using a compiler plugin, LXFI instruments the generated code to grant, check, and transfer capabilities between modules, according to the programmer's annotations. An evaluation with Linux shows that the annotations required on kernel functions to support a new module are moderate, and that LXFI is able to prevent three known privilege-escalation vulnerabilities. Stress tests of a network driver module also show that isolating this module using LXFI does not hurt TCP throughput but reduces UDP throughput by 35%, and increases CPU utilization by 2.2–3.7×.

Categories and Subject Descriptors: D.4.6 [**Operating Systems**]: Security and Protection.

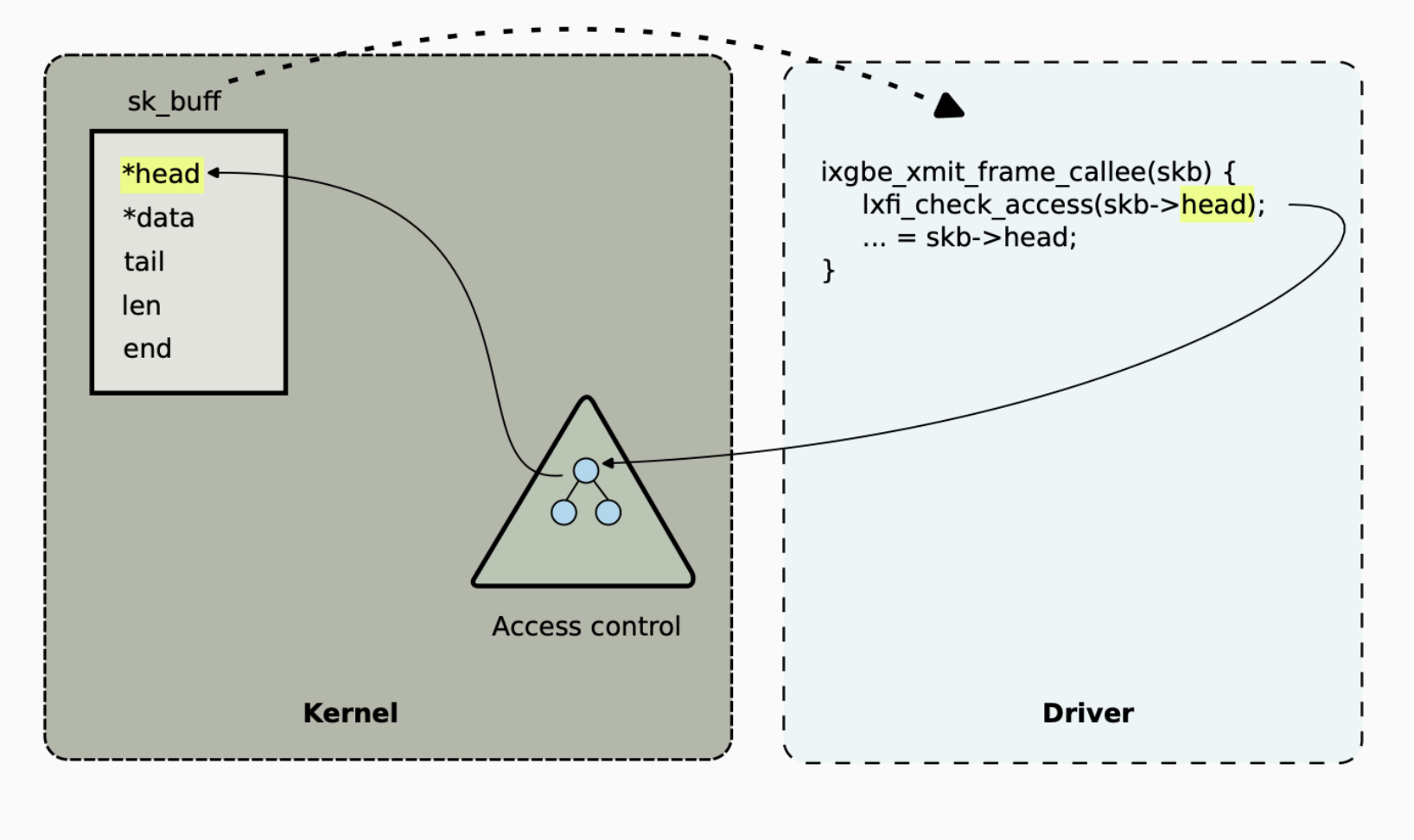
General Terms: Security.

complex, irregular kernel interfaces such as the ones found in the Linux kernel and exploited by attackers.

Previous systems such as XFI [9] have used software isolation [26] to isolate kernel modules from the core kernel, thereby protecting against a class of attacks on kernel modules. The challenge is that modules need to use support functions in the core kernel to operate correctly; for example, they need to be able acquire locks, copy data, etc., which require invoking functions in the kernel core for these abstractions. Since the kernel does not provide type safety for pointers, a compromised module can exploit some seemingly "harmless" kernel API to gain privilege. For instance, the `spin_lock_init` function in the kernel writes the value zero to a spinlock that is identified by a pointer argument. A module that can invoke `spin_lock_init` could pass the address of the user ID value in the current process structure as the spinlock pointer, thereby tricking `spin_lock_init` into setting the user ID of the current process to zero (i.e., root in Linux), and gaining root privileges.

Two recent software fault isolation systems, XFI and BGI [4], have two significant shortcomings. First, neither can deal with complex, irregular interfaces; as noted by the authors of XFI, attacks by modules that abuse an over-permissive kernel routine that a module is allowed to invoke remain an open problem [9, §6.1]. BGI tackles this problem in the context of Windows kernel drivers, which have a well-defined regular structure amenable to manual interposition on all kernel/module interactions. The Linux kernel, on the other hand, has a more complex interface that makes manual interposition

LXFI


sk_buff
*head
*data
tail
len
end
Access control
Kernel
ixgbe_xmit_frame_callee(skb) {
lxfi_check_access(skb->head);
... = skb->head;
}
Driver

Performance

Test	Throughput		CPU %	
	Stock	LXFI	Stock	LXFI
TCP_STREAM TX	836 M bits/sec	828 M bits/sec	13%	48%
TCP_STREAM RX	770 M bits/sec	770 M bits/sec	29%	64%
UDP_STREAM TX	3.1 M/3.1 M pkt/sec	2.0 M/2.0 M pkt/sec	54%	100%
UDP_STREAM RX	2.3 M/2.3 M pkt/sec	2.3 M/2.3 M pkt/sec	46%	100%
TCP RR	9.4 K Tx/sec	9.4 K Tx/sec	18%	46%
UDP RR	10 K Tx/sec	8.6 K Tx/sec	18%	40%
TCP RR (1-switch latency)	16 K Tx/sec	9.8 K Tx/sec	24%	43%
UDP RR (1-switch latency)	20 K Tx/sec	10 K Tx/sec	23%	47%

Figure 12: Performance of `netperf` benchmark with stock and LXFI enabled `e1000` driver.

- UDP traffic saturates CPU

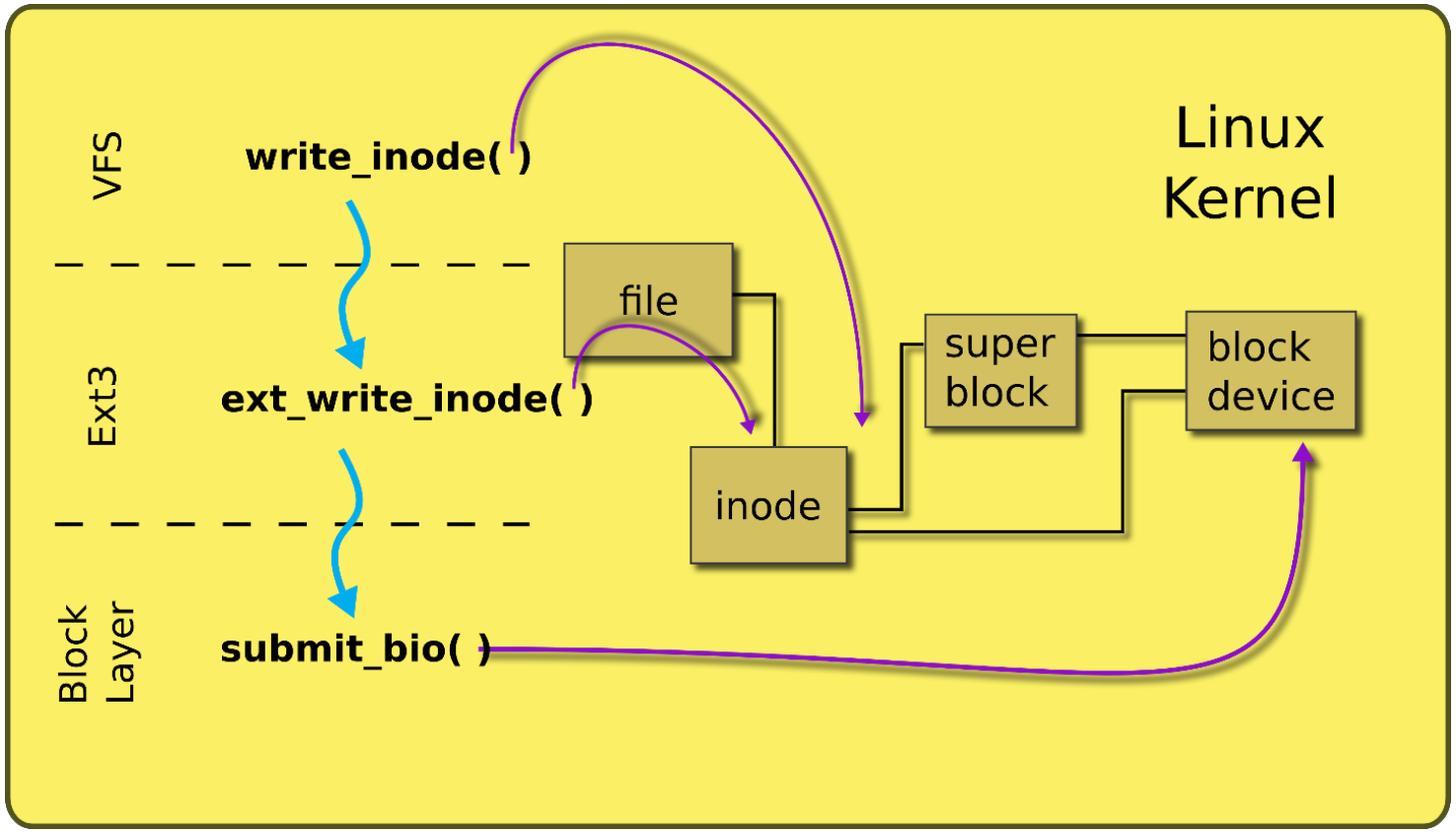
- Annotations are complex
  - They are hard to do by hand
  - Not sure they always work for data structures like spinlock

```
43 struct pci_driver {
44   int (*probe) (struct pci_dev *pcidev, ...)
45     principal(pcidev)
46     pre(copy(ref(struct pci_dev), pcidev))
47     post(if (return < 0)
48            transfer(ref(struct pci_dev), pcidev));
49 };
50
51 void skb_caps(struct sk_buff *skb) {
52   lxfi_cap_iterate(write, skb, sizeof(*skb));
53   lxfi_cap_iterate(write, skb->data, skb->len);
54 }
55
56 struct net_device_ops {
57   netdev_tx_t (*ndo_start_xmit)
58     (struct sk_buff *skb,
59      struct net_device *dev)
60     principal(dev)
61     pre(transfer(skb_caps(skb)))
62     post(if (return == -NETDEV_BUSY)
63            transfer(skb_caps(skb)))
64 };
65
66 void pci_enable_device(struct pci_dev *pcidev)
67   pre(check(ref(struct pci_dev), pcidev));
68
69 int
70 module_pci_probe(struct pci_dev *pcidev) {
71   ndev = alloc_etherdev(...);
72   lxfi_check(ref(struct pci_dev), pcidev);
73   lxfi_princ_alias(pcidev, ndev);
74   pci_enable_device(pcidev);
75   ndev->dev_ops->ndo_start_xmit = myxmit;
76   netif_napi_add(ndev, napi, my_poll_cb);
77   return 0;
78 }
```

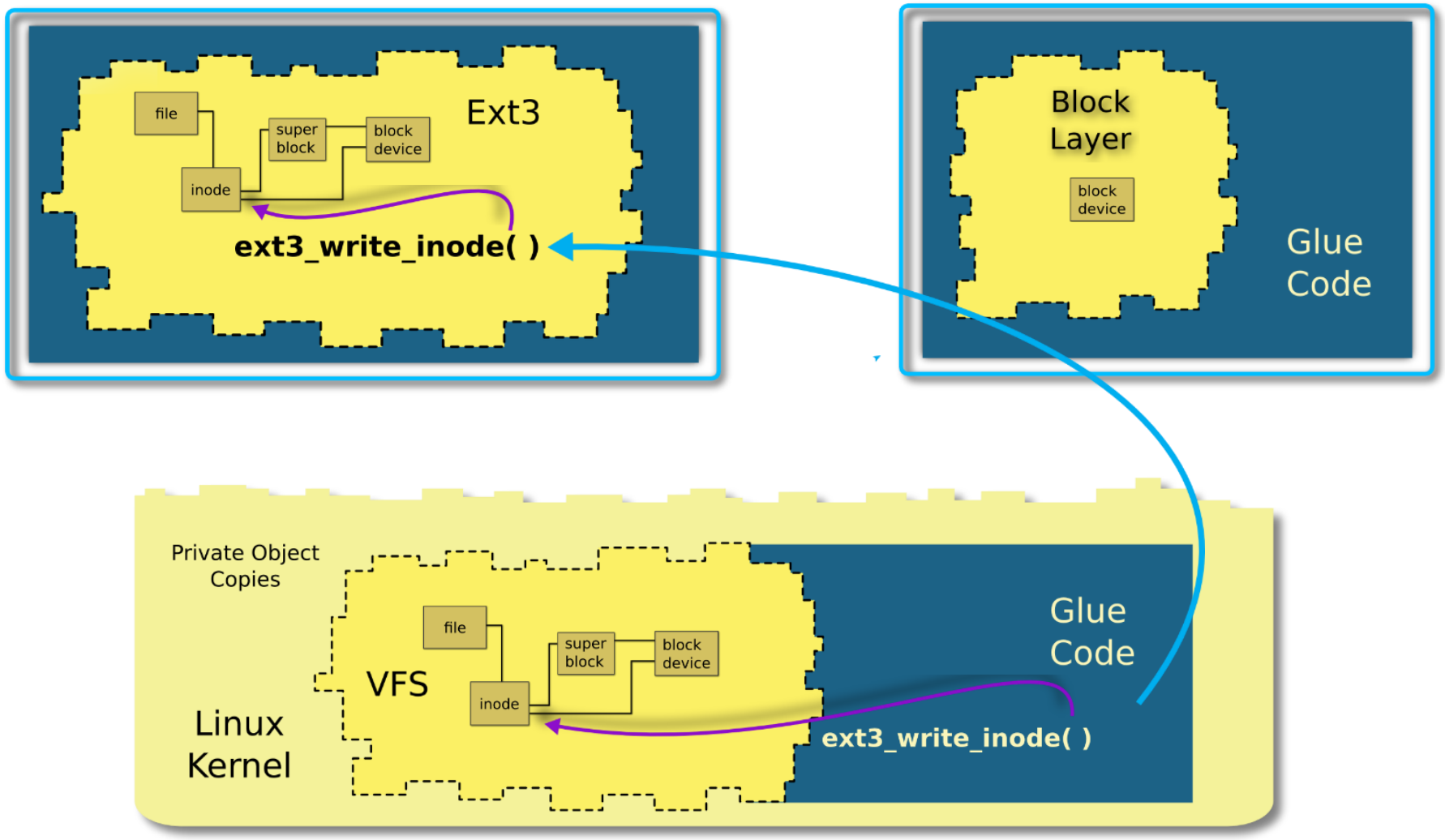
Figure 4: Annotations for parts of the API shown in Figure 1. The annotations follow the grammar shown in Figure 2. Annotations and added code are underlined.

LXDs and LVDs: Hardware isolation

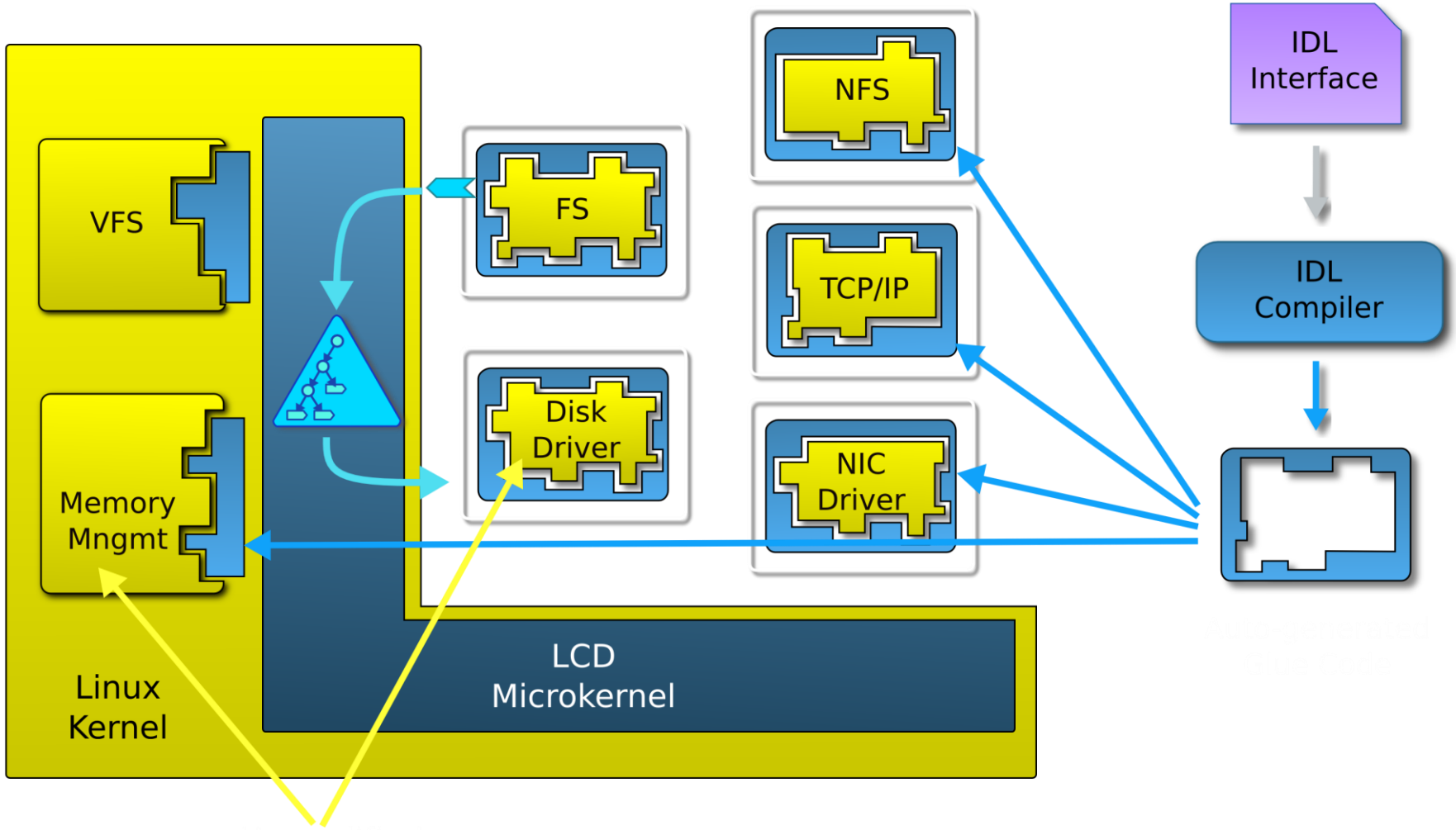
Commodity OS: Shared Object Space



LCDs: Isolated object spaces



Lightweight capability domains



Driver isolation architecture

- Separate memory space

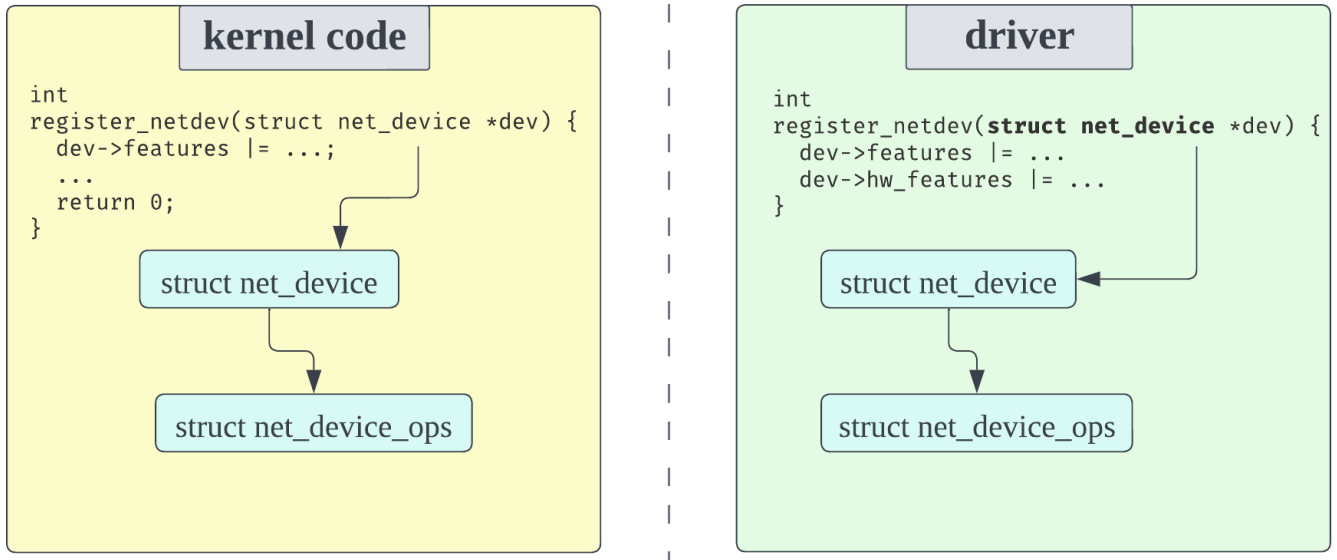
kernel code

```
int
register_netdev(struct net_device *dev) {
  dev->features |= ...;
  ...
  return 0;
}
```

driver

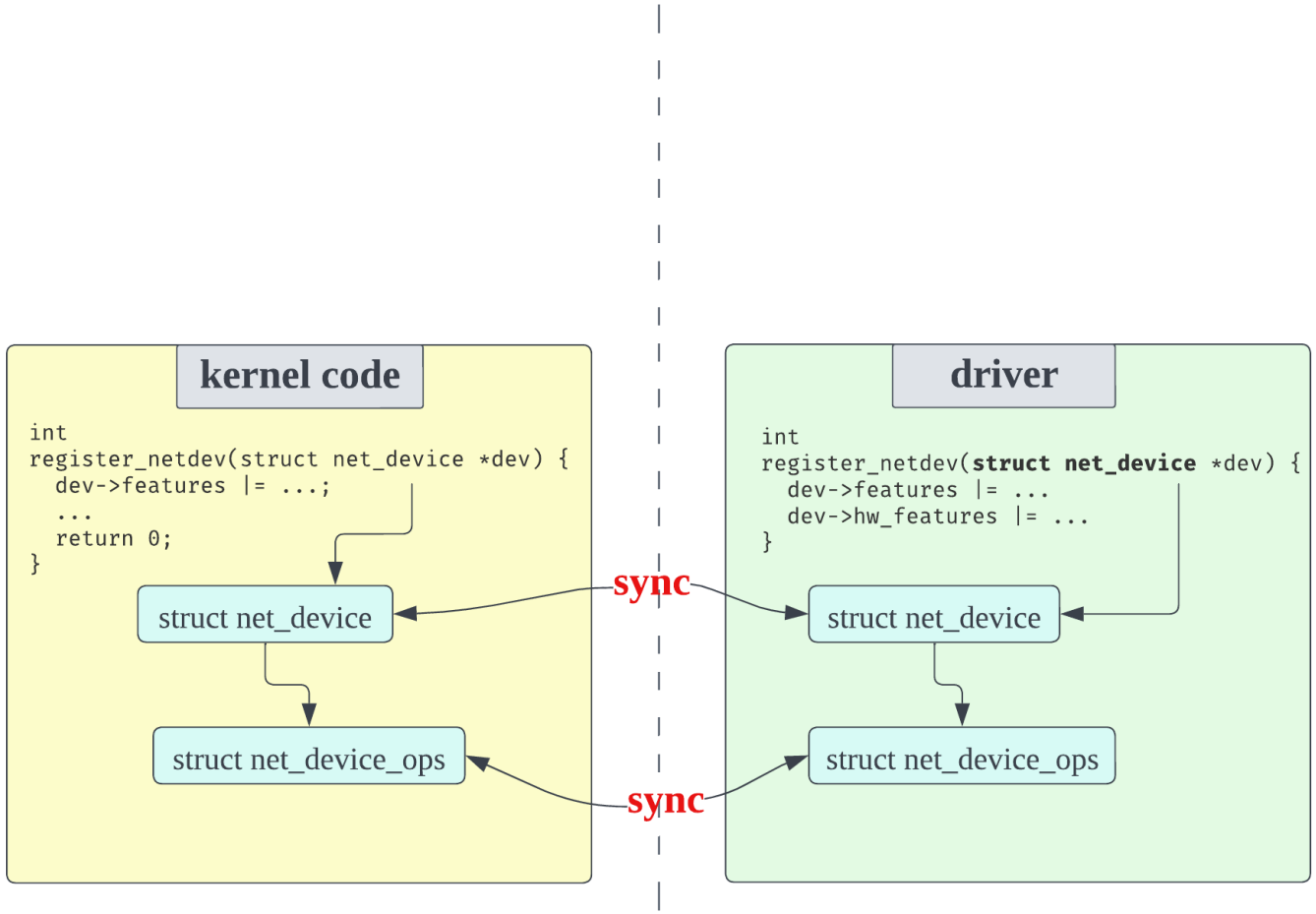
```
int
register_netdev(struct net_device *dev) {
  dev->features |= ...
  dev->hw_features |= ...
}
```

Driver isolation architecture



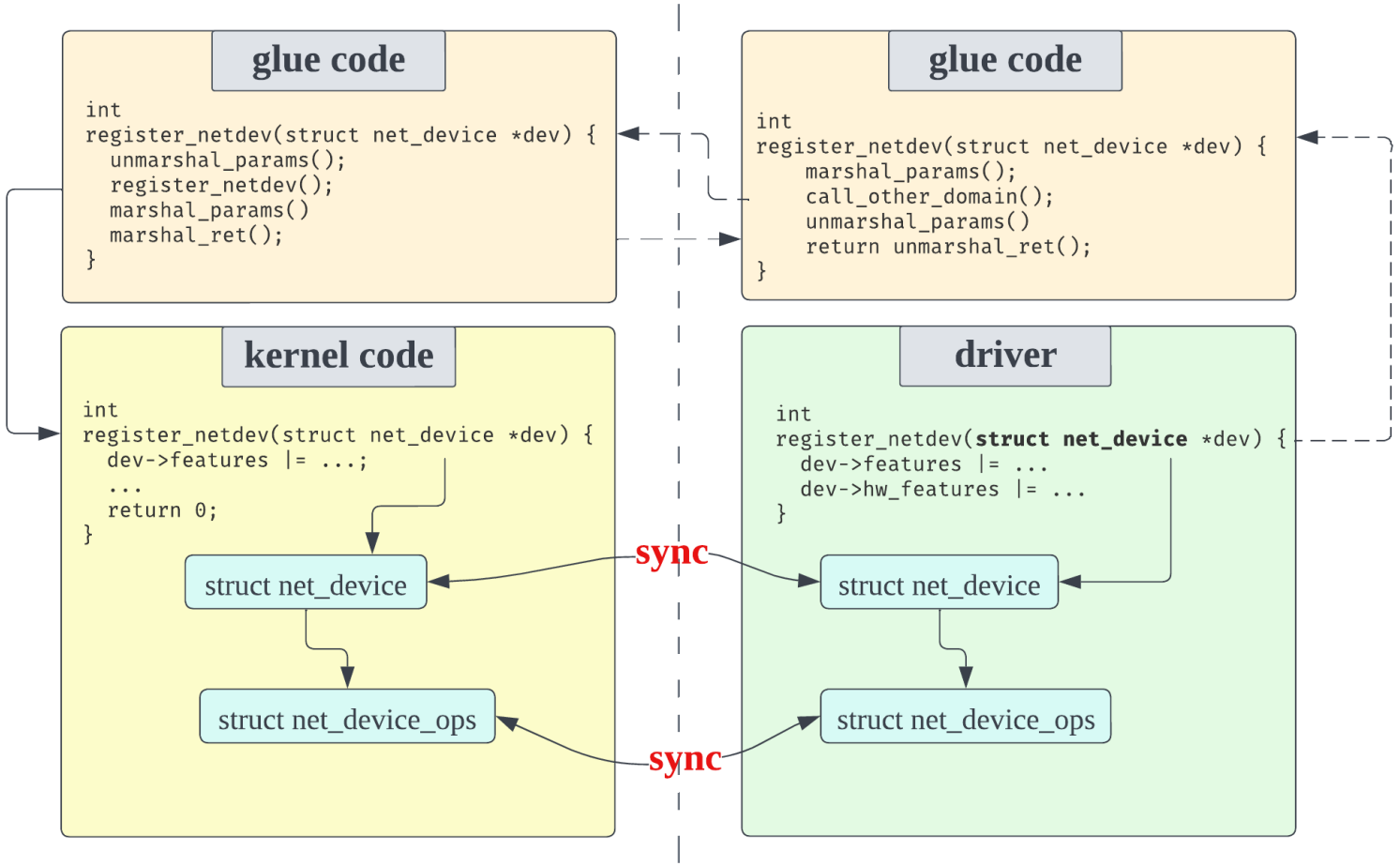
- Separate memory space
- Two copies of object hierarchies

Driver isolation architecture



- Separate memory space
  - Two copies of object hierarchies
  - Keep them synchronized

Driver isolation architecture

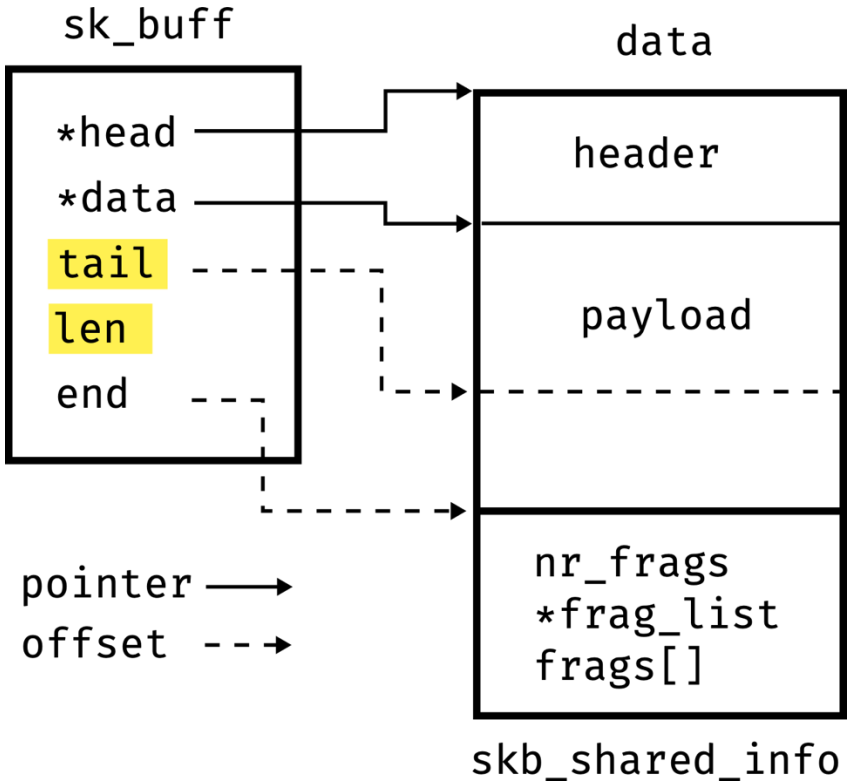


- Separate memory space
  - Two copies of object hierarchies
  - Keep them synchronized

- Glue code
  - Marshal/unmarshal params
  - Interface definition language (IDL) spec
  - Generated with IDL compiler

Semantic complexity

netdev_tx_t
ixgbe_xmit_frame(struct sk_buff *skb, ...)

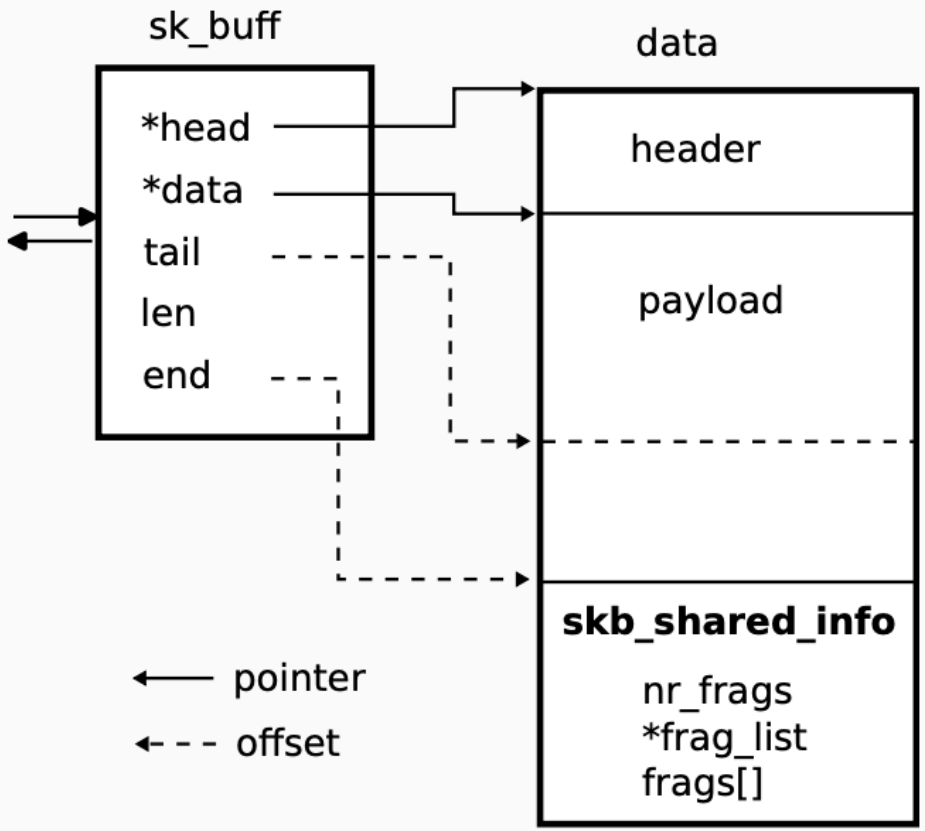


- Represents a network packet
- Has 66 fields (5 pointers)
- 3,132 fields (1,214 pointers) are recursively reachable
- But only a small subset is accessed by both kernel and driver (shared)
  - 8 shared fields for this API

IDL example

Ixgbe network driver

```
module ixgbe {
    rpc int ndo_start_xmit(projection net_device *dev, projection sk_buff *skb) {
      projection <struct net_device> dev {
        ...
      };
      projection <struct sk_buff> skb {
        void *data;
        u32 len;
                    ...
      };
    }
}
```




```
rpc_ptr int ndo_xmit(
  projection sk_buff [in] *skb,
  projection net_device  *netdev)
{
  projection <struct sk_buff> sk_buff {
    unsigned int [in,out] len;
    unsigned int [in] data_len;
    unsigned short [in] mac_len;
    unsigned short [in] queue_mapping;
    unsigned char [in] cloned : 1;
    unsigned int [in] priority;
    unsigned short [in] vlan_tci;
    unsigned short [in] protocol;
    array<unsigned int, 0> [in,out]
    ↪  headers_end;
    unsigned int [in,out] tail;
    unsigned int [in] end;
    unsigned char* [in] head;
    unsigned char* [in] data;
  }
  projection <struct net_device>
  ↪  net_device {
  }
}
```

Lightweight eXecution Domains (LXDs)

USENIX ATC'19

LXDs: Towards Isolation of Kernel Subsystems

Vikram Narayanan
University of California, Irvine

Abhiram Balasubramanian*
University of Utah

Charlie Jacobsen*
University of Utah

Sarah Spall†
University of Utah

Scott Bauer‡
University of Utah

Michael Quigley§
University of Utah

Aftab Hussain
University of California, Irvine

Abdullah Younis¶
University of California, Irvine

Junjie Shen
University of California, Irvine

Moinak Bhattacharyya
University of California, Irvine

Anton Burtsev
University of California, Irvine

Abstract

Modern operating systems are monolithic. Today, however, lack of isolation is one of the main factors undermining security of the kernel. Inherent complexity of the kernel code and rapid development pace combined with the use of unsafe, low-level programming language results in a steady stream of errors. Even after decades of efforts to make commodity kernels more secure, i.e., development of numerous static and dynamic approaches aimed to prevent exploitation of most common errors, several hundreds of serious kernel vulnerabilities are reported every year. Unfortunately, in a monolithic kernel a single exploitable vulnerability potentially provides an attacker with access to the entire kernel.

Modern kernels need isolation as a practical means of con-

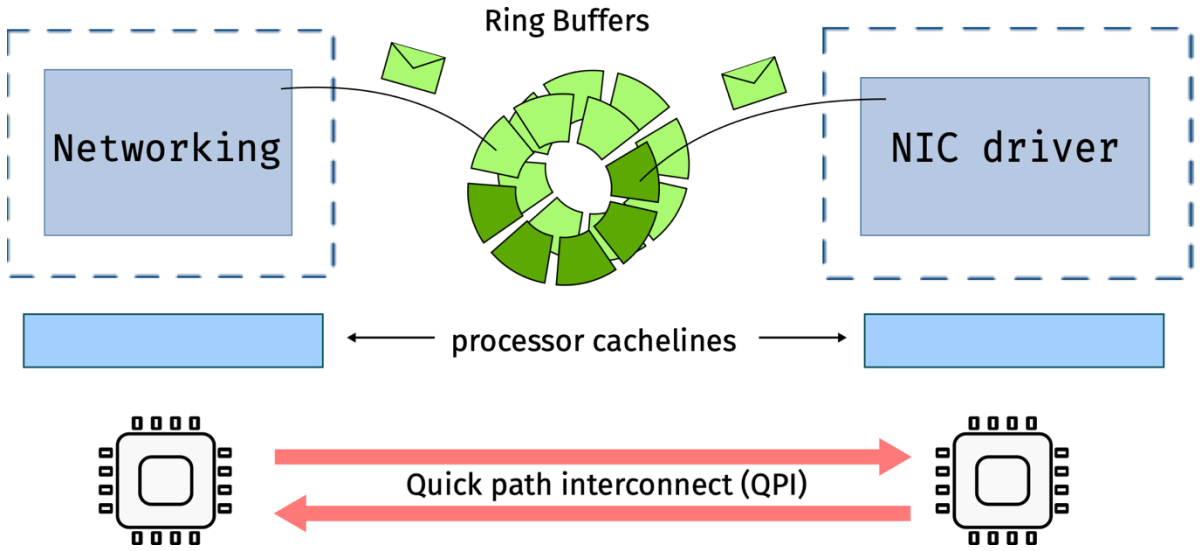
and inherent complexity (typical kernel code adheres to multiple allocation, synchronization, access control, and object lifetime conventions) bugs and vulnerabilities are routinely introduced into the kernel code. In 2018, the Common Vulnerabilities and Exposures database lists 176 Linux kernel vulnerabilities that allow for privilege escalation, denial-of-service, and other exploits [20]. This number is the lowest across several years [19].

Even though a number of static and dynamic mechanisms have been invented to protect execution of the low-level kernel code, e.g., modern kernels deploy stack guards [18], address space layout randomization (ASLR) [45], and data execution prevention (DEP) [72], attackers come up with new ways to bypass these protection mechanisms [8, 35, 45, 49, 51, 57–59,

Vikram Narayanan, Abhiram Balasubramanian, Charlie Jacobsen, Sarah Spall, Scotty Bauer, Michael Quigley, Aftab Hussain et al.
"LXDs: Towards Isolation of Kernel Subsystems." In *USENIX Annual Technical Conference*, pp. 269-284. 2019.

LXDs: Isolation boundary

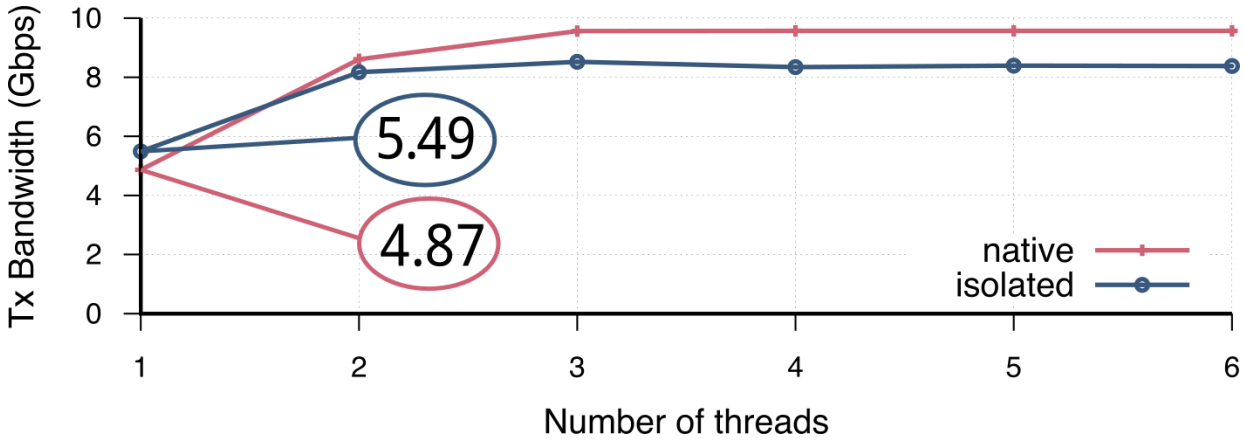
- Asynchronous call/reply mechanism
  - Caller/callee occupies two different cores
  - Leverage cache-coherence between cores to pass messages
  - Cross-core IPC (~380 cycles)



Vikram Narayanan, Abhiram Balasubramanian, Charlie Jacobsen, Sarah Spall, Scott Bauer, Michael Quigley, Aftab Hussain et al. "{LXDs}: Towards Isolation of Kernel Subsystems." In 2019 USENIX Annual Technical Conference (USENIX ATC 19), pp. 269-284. 2019.

Ixgbe performance benchmark - Tx Bandwidth

- Single thread, isolated is 12% faster
  - Benefit of burning one additional core
- For 2-6 application threads
  - Within 6-13% of the native performance



Limitations

- Asynchronous execution model is cumbersome
  - Hard to debug
    - Critical sections, error handling, etc.
  - Burn more cores

Lightweight Virtualization Domains (LVDs)

VEE'20

Lightweight Kernel Isolation with Virtualization and VM Functions

Vikram Narayanan
University of California, Irvine

Yongzhe Huang
Pennsylvania State University

Gang Tan
Pennsylvania State University

Trent Jaeger
Pennsylvania State University

Anton Burtsev
University of California, Irvine

Abstract

Commodity operating systems execute core kernel subsystems in a single address space along with hundreds of dynamically loaded extensions and device drivers. Lack of isolation within the kernel implies that a vulnerability in any of the kernel subsystems or device drivers opens a way to mount a successful attack on the entire kernel.

Historically, isolation within the kernel remained prohibitive due to the high cost of hardware isolation primitives. Recent CPUs, however, bring a new set of mechanisms. Extended page-table (EPT) switching with VM functions and memory protection keys (MPKs) provide memory isolation and invocations across boundaries of protection domains with overheads comparable to system calls. Unfortunately, neither MPKs nor EPT switching provide architectural support for isolation of privileged ring 0 kernel code, i.e., control of privileged instructions and well-defined entry points to securely restore state of the system on transition between isolated domains.

approach by developing isolated versions of several device drivers in the Linux kernel.

ACM Reference Format:
Vikram Narayanan, Yongzhe Huang, Gang Tan, Trent Jaeger, and Anton Burtsev. 2020. Lightweight Kernel Isolation with Virtualization and VM Functions. In *Proceedings of (VEE '20).* ACM, New York, NY, USA, 15 pages. https://doi.org/10.1145/3381052.3381328

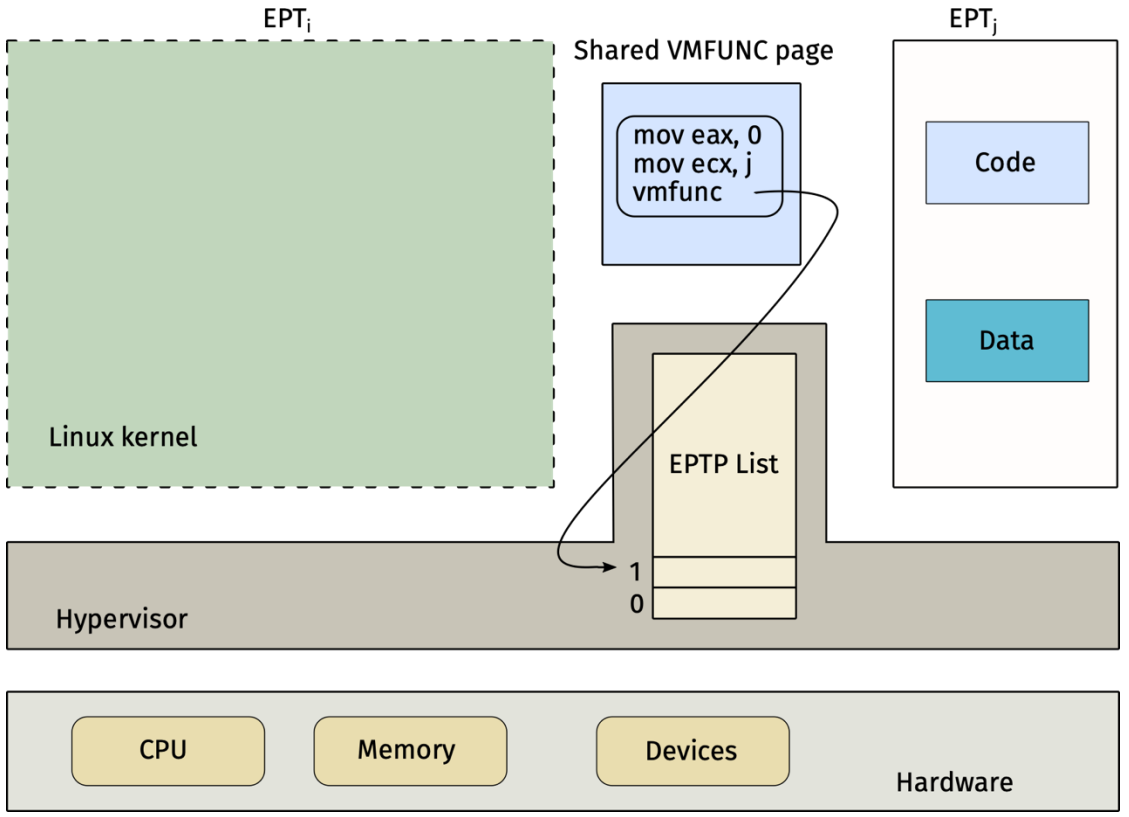
1 Introduction

Despite many arguments for running kernel subsystems in separate protection domains over the years, commodity operating systems remain monolithic. Today, the lack of isolation within the operating system kernel is one of the main factors undermining its security. While the core kernel is relatively stable, the number of kernel extensions and device drivers is growing with every hardware generation (a modern Linux kernel contains around 8,867 device drivers [3], with around 80-130 drivers running on a typical system). Developed by

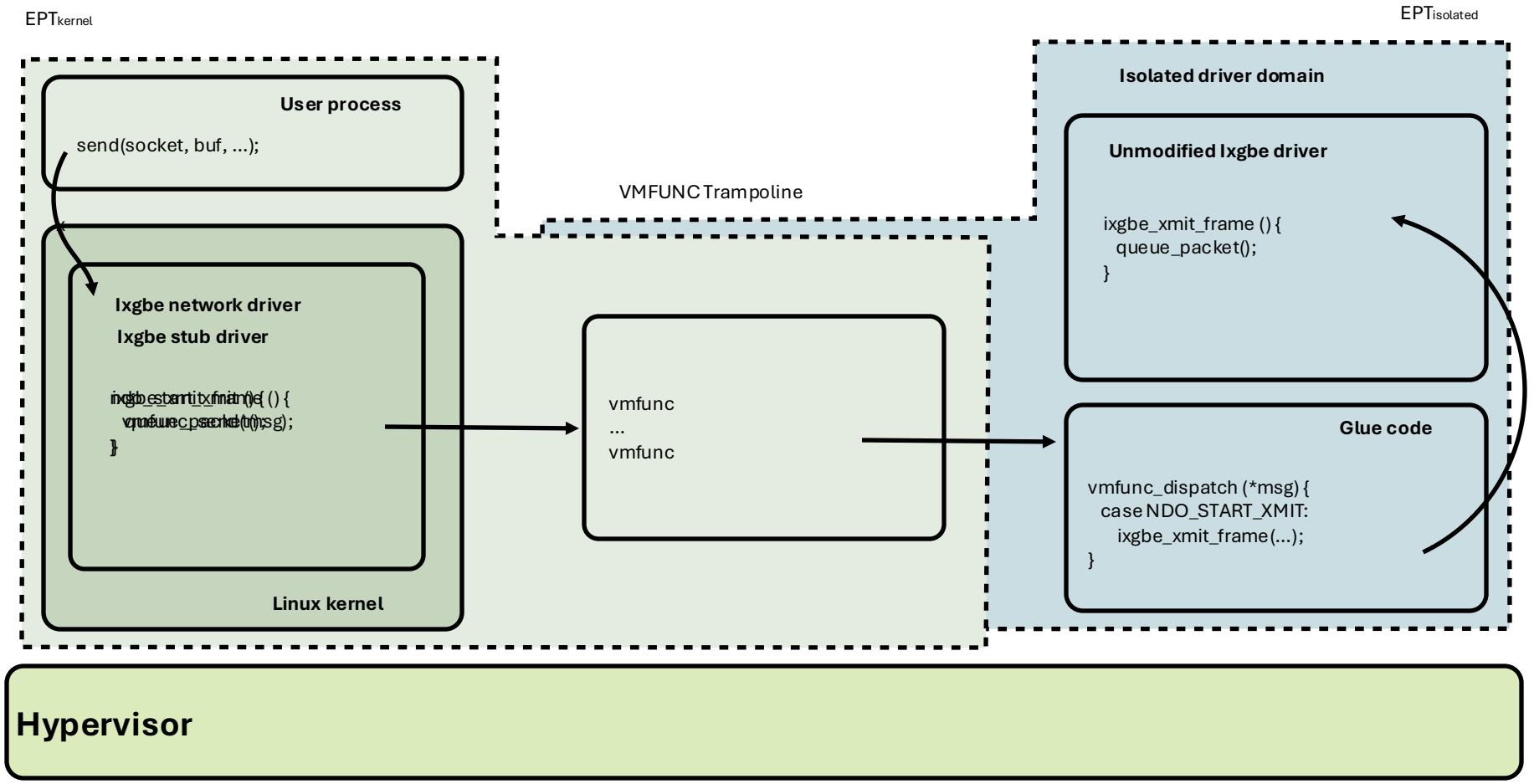
Vikram Narayanan, Yongzhe Huang, Gang Tan, Trent Jaeger, and Anton Burtsev. "Lightweight kernel isolation with virtualization and vm functions."
In Proceedings of the 16th ACM SIGPLAN/SIGOPS International Conference on **Virtual Execution Environments**, pp. 157-171. 2020.

VM Functions

- Switch from one extended page table (EPT) to another
- List of pre-configured EPTP pointers
- Low-overhead (~400 cycles)



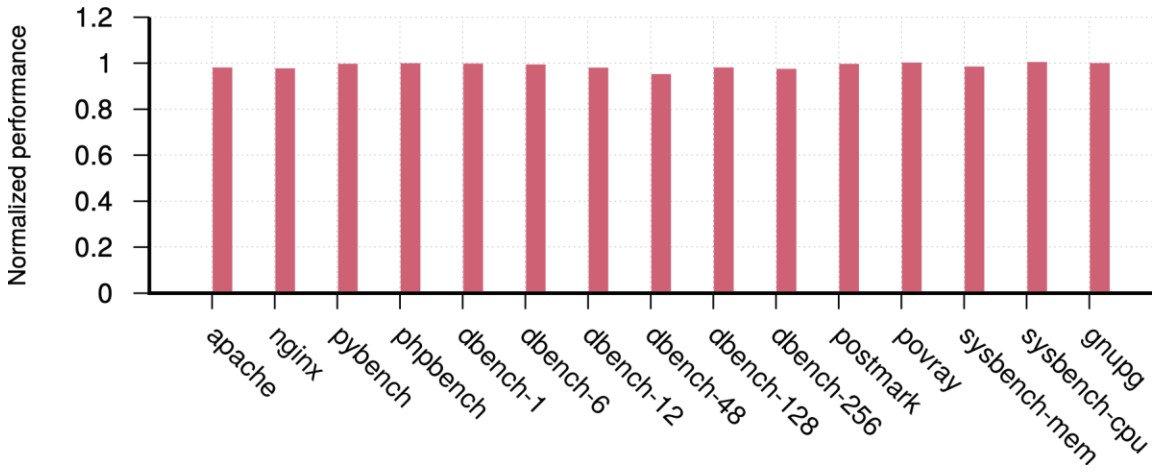
Architecture



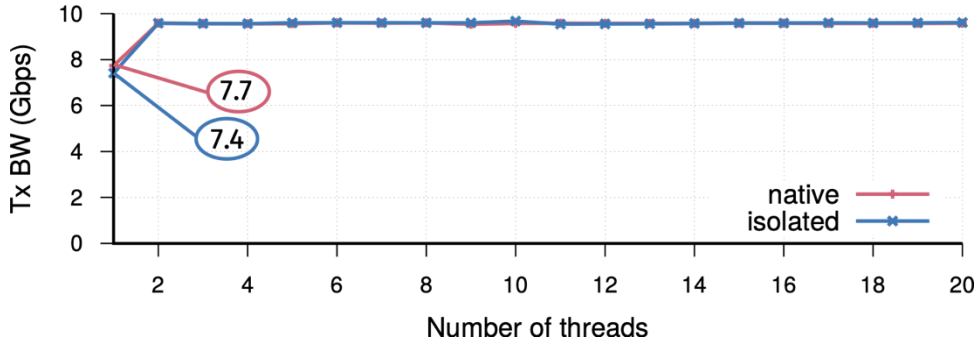
System Performance

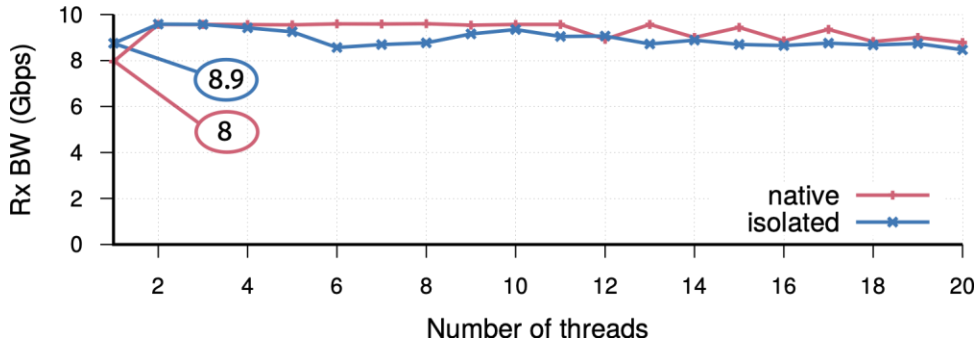
Phoenix Test Suite (PTS)

- 2-5% overhead
  - apache, nginx, sys bench (2%)
  - dbench (2-5%)



- Iperf2 benchmarks





KSplit - Static analysis for decomposition

OSDI'22

KSplit: Automating Device Driver Isolation

Yongzhe Huang*[1], Vikram Narayanan*[2], David Detweiler[2], Kaiming Huang[1], Gang Tan[1], Trent Jaeger[1], and Anton Burtsev[2,3]

[1]Pennsylvania State University
[2]University of California, Irvine
[3]University of Utah

Abstract

Researchers have shown that recent CPU extensions support practical, low-overhead driver isolation to protect kernels from defects and vulnerabilities in device drivers. With performance no longer being the main roadblock, the complexity of isolating device drivers has become the main challenge. Device drivers and kernel extensions are developed in a shared memory environment in which the state shared between the kernel and the driver is mixed in a complex hierarchy of data structures, making it difficult for programmers to ensure that the shared state is synchronized correctly. In this paper, we present KSplit, a new framework for isolating unmodified device drivers in a modern, full-featured kernel. KSplit performs automated analyses on the unmodified source code of the kernel and the driver to: 1) identify the state shared between the kernel and driver and 2) to compute the synchronization requirements for this shared state for efficient isolation. While some kernel idioms present ambiguities that cannot be resolved automatically at present, KSplit classifies most ambiguous pointers and identifies ones requiring manual

and device drivers is large and continues to grow. A modern Linux 5.12 kernel contains around 8,960 device drivers that account for 67.7% of the kernel source [3], a number that has nearly doubled since 2013. With a rate of 80,000 commits a year, defects and vulnerabilities are an inherent part of the fast growing and evolving driver codebase.

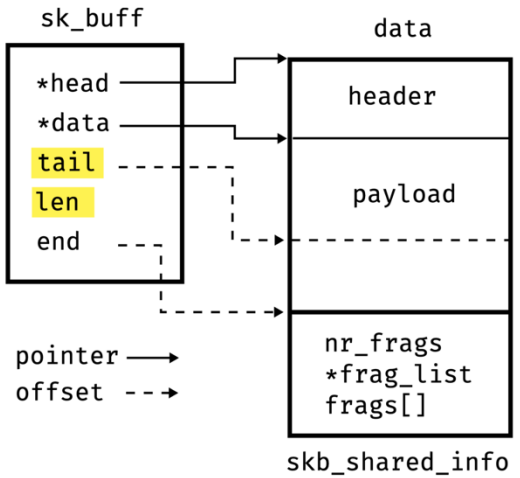
The recent availability of hardware features for efficient isolation [1, 5] and system support that leverages such features [40, 43, 47, 61, 63, 82] have made low-overhead device isolation frameworks practical [66, 68]. The upcoming hardware extensions, e.g., native page-granularity support for isolation of kernel code [5], and 16 byte-granularity isolation with memory tagging extensions (MTE) [1], which are key for enabling low-overhead software fault isolation (SFI) implementations [53], will reduce isolation overheads even more.

Despite the availability of low-overhead isolation mechanisms, the task of isolating existing driver code remains challenging. For decades, device drivers and kernel extensions have been developed in the shared memory environment of a monolithic kernel, where they freely exchange references to

Yongzhe Huang*, **Vikram Narayanan*,** David Detweiler, Kaiming Huang, Gang Tan, Trent Jaeger, and Anton Burtsev. "{KSplit}: Automating Device Driver Isolation." In 16th USENIX Symposium on **Operating Systems Design and Implementation (OSDI 22)**, pp. 613-631. 2022.

Semantic complexity

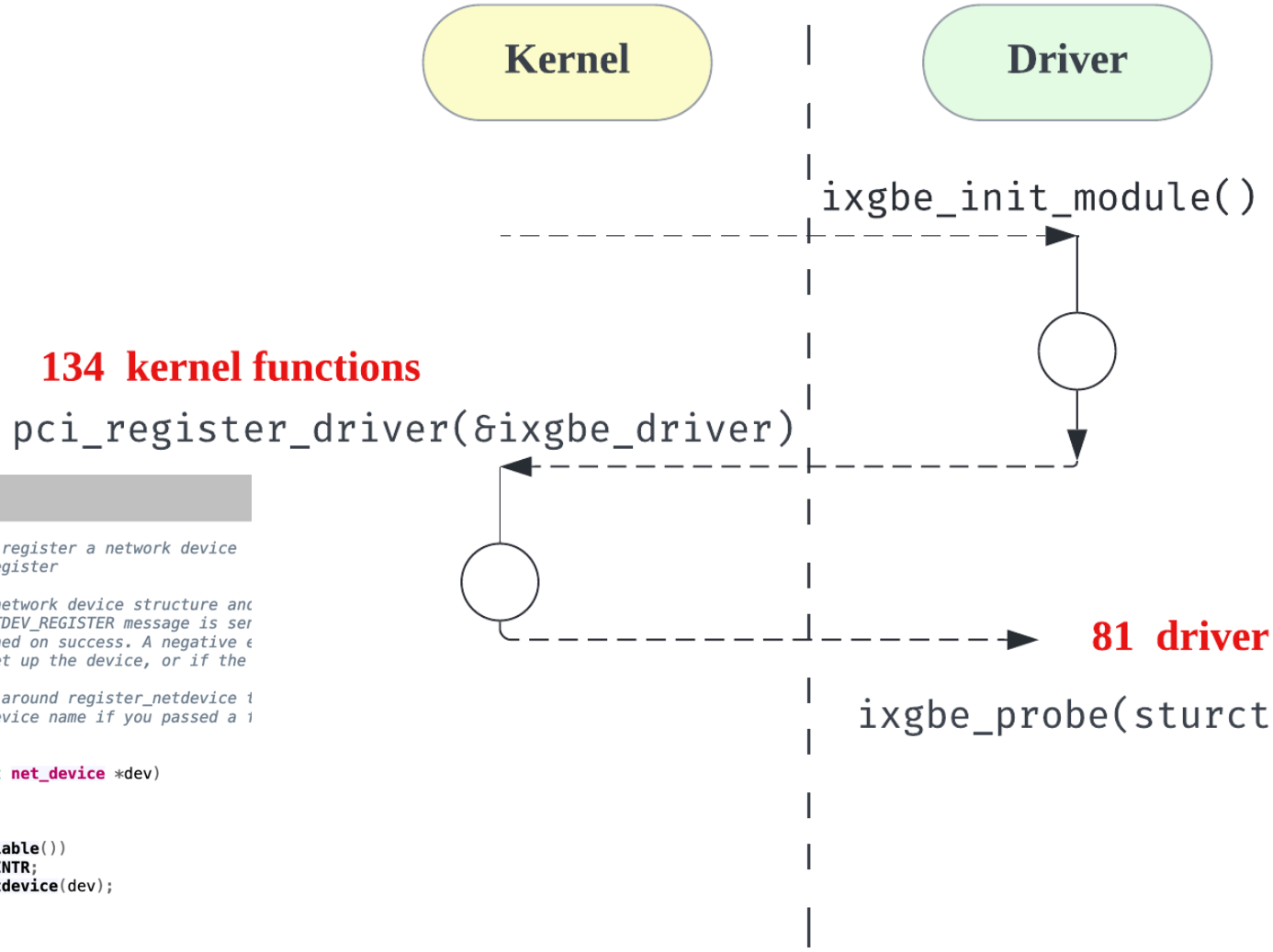
```
netdev_tx_t
ixgbe_xmit_frame(struct sk_buff *skb, …)
```



- Represents a network packet
- Has 66 fields (5 pointers)
- 3,132 fields (1,214 pointers) are recursively reachable
- But only a small subset is accessed by both kernel and driver (shared)
  - 8 shared fields for this API

Manually specifying the IDL for data synchronization between domains has become the major challenge

Challenge: Large interface boundary



/ drivers / net / ethernet / intel / ixgbe / ixgbe_main.c

```
10400  static const struct net_device_ops ixgbe_netdev_ops = {
10401          .ndo_open               = ixgbe_open,
10402          .ndo_stop               = ixgbe_close,
10403          .ndo_start_xmit         = ixgbe_xmit_frame,
10404          .ndo_set_rx_mode        = ixgbe_set_rx_mode,
10405          .ndo_validate_addr      = eth_validate_addr,
10406          .ndo_set_mac_address    = ixgbe_set_mac,
10407          .ndo_change_mtu         = ixgbe_change_mtu,
10408          .ndo_tx_timeout         = ixgbe_tx_timeout,
10409          .ndo_set_tx_maxrate     = ixgbe_tx_maxrate,
10410          .ndo_vlan_rx_add_vid    = ixgbe_vlan_rx_add_vid,
10411          .ndo_vlan_rx_kill_vid   = ixgbe_vlan_rx_kill_vid,
10412          .ndo_eth_ioctl          = ixgbe_ioctl,
10413          .ndo_set_vf_mac         = ixgbe_ndo_set_vf_mac,
10414          .ndo_set_vf_vlan        = ixgbe_ndo_set_vf_vlan,
10415          .ndo_set_vf_rate        = ixgbe_ndo_set_vf_bw,
10416          .ndo_set_vf_spoofchk    = ixgbe_ndo_set_vf_spoofchk,
10417          .ndo_set_vf_link_state  = ixgbe_ndo_set_vf_link_state,
10418          .ndo_set_vf_rss_query_en = ixgbe_ndo_set_vf_rss_query_en,
10419          .ndo_set_vf_trust       = ixgbe_ndo_set_vf_trust,
10420          .ndo_get_vf_config      = ixgbe_ndo_get_vf_config,
10421          .ndo_get_vf_stats       = ixgbe_ndo_get_vf_stats,
10422          .ndo_get_stats64        = ixgbe_get_stats64,
10423          .ndo_setup_tc           = __ixgbe_setup_tc,
```

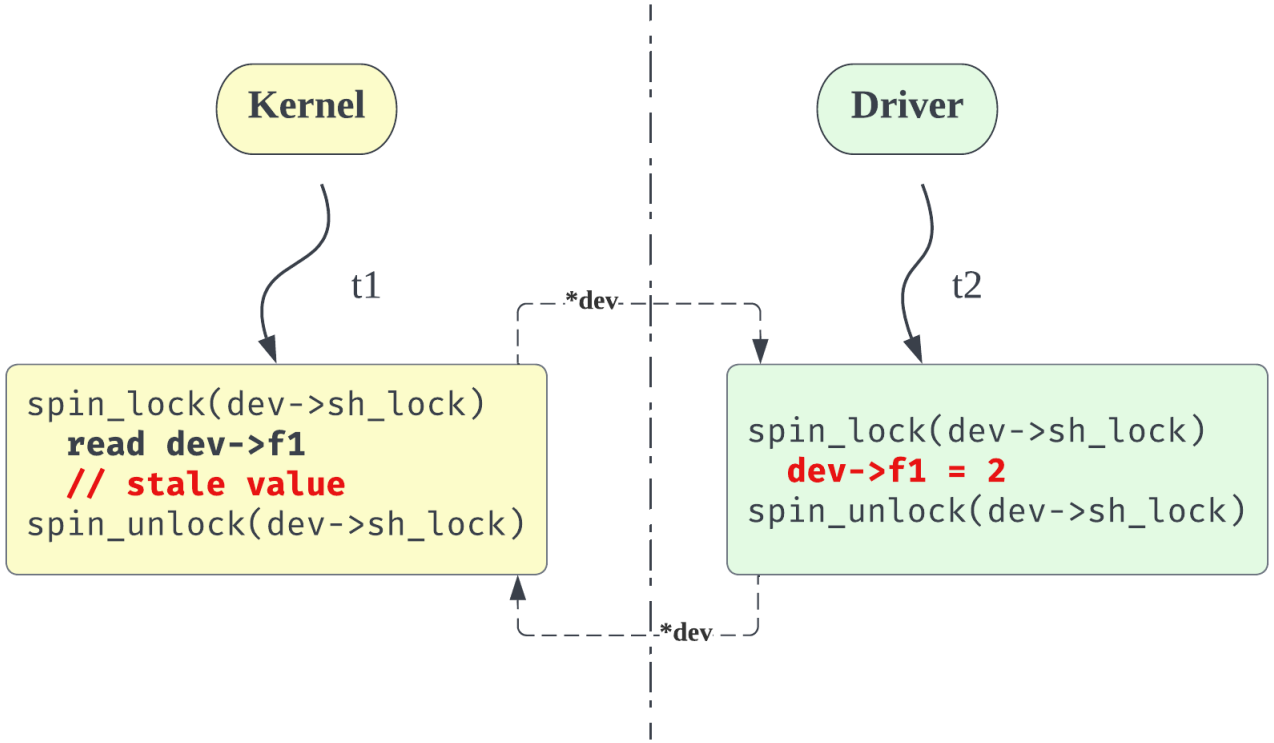
/ net / core / dev.c

```
10128  /**
10129   *      register_netdev - register a network device
10130   *      @dev: device to register
10131   *
10132   *      Take a completed network device structure and
10133   *      interfaces. A %NETDEV_REGISTER message is sen
10134   *      chain. 0 is returned on success. A negative e
10135   *      on a failure to set up the device, or if the
10136   *
10137   *      This is a wrapper around register_netdevice t
10138   *      and expands the device name if you passed a t
10139   *      alloc_netdev.
10140   */
10141  int register_netdev(struct net_device *dev)
10142  {
10143          int err;
10144
10145          if (rtnl_lock_killable())
10146                  return -EINTR;
10147          err = register_netdevice(dev);
10148          rtnl_unlock();
10149          return err;
10150  }
10151  EXPORT_SYMBOL(register_netdev);
```

Challenge: Low-level kernel/C idioms

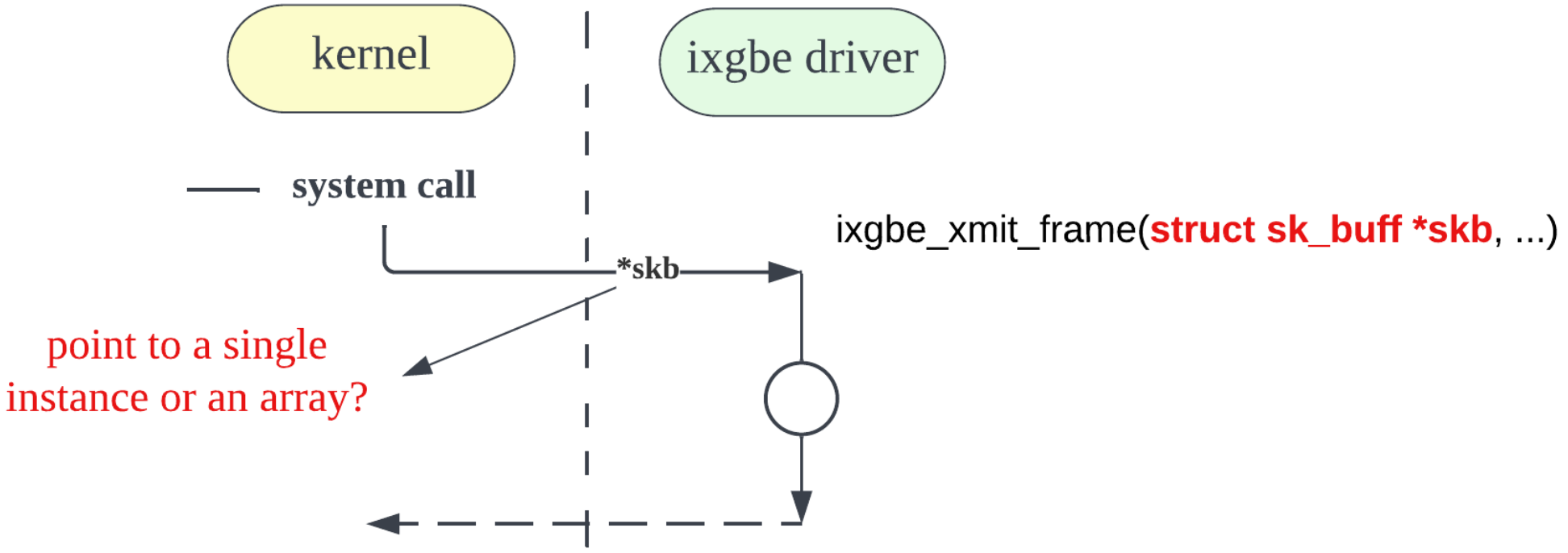
- Pointers
  - Singleton, array
  - Linked list
  - Collocated data structures
- Sized and sentinel arrays
- Special pointers (e.g., __user, __iomem)
- Tagged unions
- Return error as ptr (e.g., ERR_PTR)

Challenge: Concurrency primitives



- spin/mutex lock
- driver specific lock, e.g., rtnl_lock
- atomic operations, e.g., set_bit
- read-copy update (RCU)
- sequential lock

Challenge: Pointers



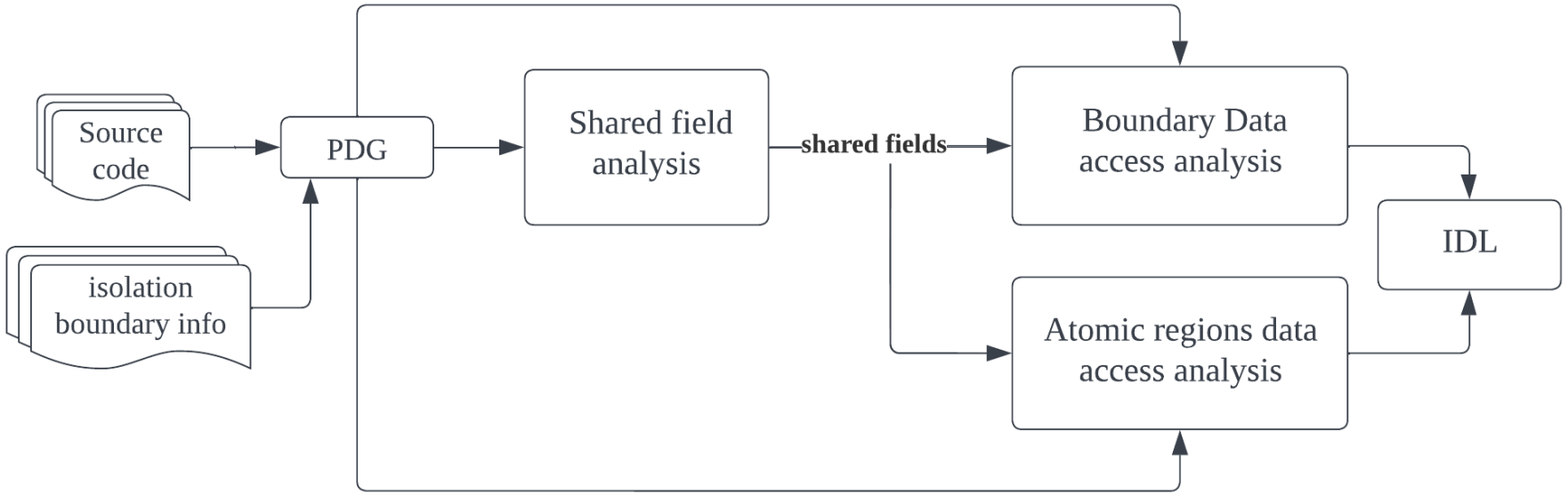
KSplit goals

- Build a set of static analyses to generate the IDL automatically (mostly) to
  - Isolate the complete driver
  - Identify shared/private data on the large interface boundary
  - Ensure each domain has the updated copy of the data structure
  - Identify marshaling requirements for the low-level kernel idioms
  - Identify atomic regions that access shared data
- Prior work
  - *Microdrivers* (isolated the control plane of the driver)

KSplit design choices

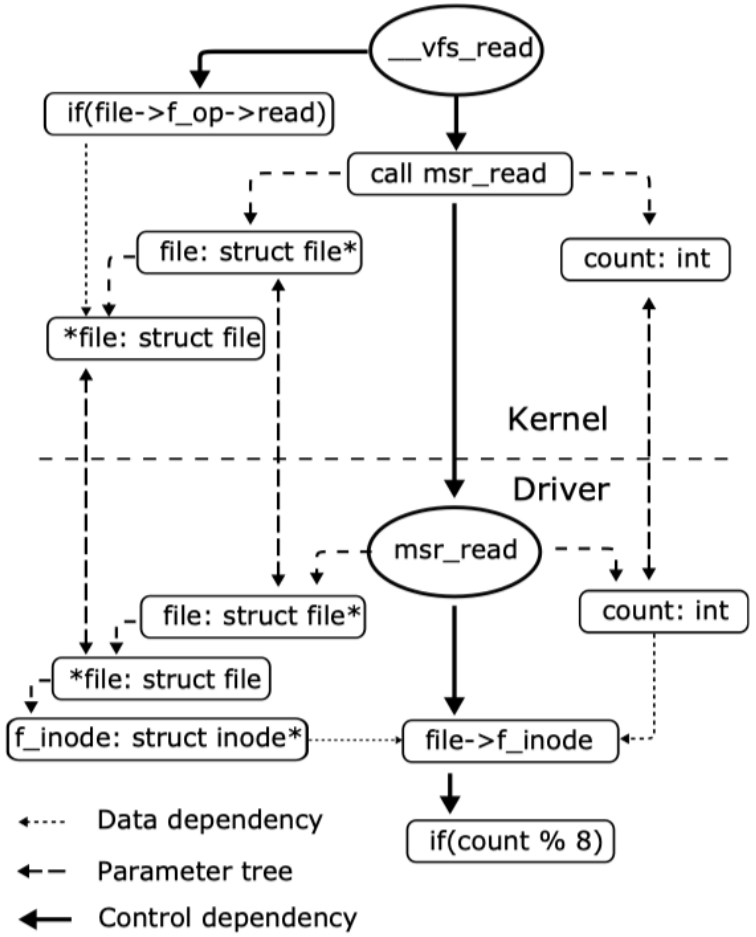
- Kernel is huge
  - Identify the relevant kernel code that the driver interacts with
- Aim to detect all shared data (sound)
  - We might classify some private data as shared
- Aim to infer marshaling requirements for low-level idioms
  - Provide warning for the cases that we cannot infer
- Aim to infer marshaling requirements for shared critical sections
  - Hypothesis: There are not many shared critical sections

KSplit workflow



- **Input**: source code of kernel and target isolated driver
- **Output**: IDL file that specifies the communication interfaces and data synchronization requirements

Program Dependence Graph



- **PDG**: represents program dependencies
  - inter-procedural pointer alias relations
  - field-sensitive
  - data dependencies
  - control dependencies/flow

Shen Liu, Gang Tan, and Trent Jaeger. "PtrSplit: Supporting general pointers in automatic program partitioning." In Proceedings of the 2017 ACM SIGSAC Conference on **Computer and Communications Security**, pp. 2359-2371. 2017.

Did we solve the problem?

Evolving Operating Systems Towards Secure Kernel-Driver Interfaces

Anton Burtsev
University of Utah

Vikram Narayanan
University of Utah

Yongzhe Huang
Pennsylvania State University

Kaiming Huang
Pennsylvania State University

Gang Tan
Pennsylvania State University

Trent Jaeger
Pennsylvania State University

Abstract

Our work explores the challenge of developing secure kernel-driver interfaces designed to protect the kernel from isolated kernel extensions. We first analyze a range of possible attack vectors that exist in current isolation frameworks. Then, we suggest a new approach to building secure isolation boundaries centered around ideas that originate in safe operating systems: isolation of heaps and single ownership.

CCS Concepts

• **Security and privacy → Operating systems security**.

ACM Reference Format:
Anton Burtsev, Vikram Narayanan, Yongzhe Huang, Kaiming Huang, Gang Tan, and Trent Jaeger. 2023. **Evolving Operating Systems Towards Secure Kernel-Driver Interfaces. In *Workshop on Hot Topics in Operating Systems (HotOS '23), June 22–24, 2023, Providence, RI, USA.* ACM, New York, NY, USA, 8 pages.** https://doi.org/10.1145/3593856.3595914

1 Introduction

Extensions (MTE) [2, 5], which can potentially enable low-overhead bounds checks and zero-copy exchange of data. Moreover, both ARM and x86 provide support for control flow integrity (CFI) [29] and stack protection [1, 28]. Finally, safe programming languages like Rust are becoming first-class citizens in modern systems [15, 43].

In response to lowering overheads of hardware and software isolation, a range of projects started to explore techniques for isolating legacy systems, many targeting isolation of the kernel subsystems like device drivers [9, 30, 31, 34, 36]. Furthermore, recent static analysis techniques demonstrated largely automated isolation of the kernel code [17]. With breakthroughs addressing two key isolation challenges—performance and complexity—it is likely that isolation will soon find its way into modern kernels.

A natural question, however, is what kind of security guarantees are achieved by current isolation frameworks? Unfortunately, even using the most advanced isolation boundary approaches that enforce temporal memory safety across the isolation boundary [30, 31], the kernel can be attacked in numerous ways.

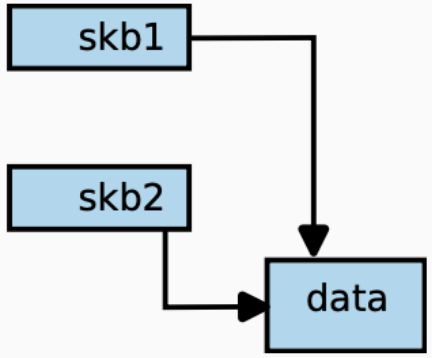
Out-of-bounds accesses

- Trigger an out-of-bounds access through a field that controls offsets into memory

```
// driver update
skb_frag_size_sub(frag, pull_len);
frag->page_offset  += pull_len;
skb->data_len -= pull_len;
```

```
// kernel read
csum2 =
csum_partial_copy_nocheck(vaddr +
   frag->page_offset  +
   offset - start, to,
   copy, 0);
```

Pointer aliases



- Confuse the kernel to trigger a use-after-free or double-free

```
// driver
skb1->data = skb2->data;


// kernel
kfree_skb(skb1);
...
// use after free
... = *skb2->data
```

Lifetimes

- Trigger deallocation
  - `struct netdev *dev`
  - Lifetime of the driver
- Confuse reference counting
  - `struct sk_buff *skb`
  - Reference counted

```
// driver
free_netdev(dev);

// driver
// decrement ref count
consume_skb(skb);
...

// decrement again
consume_skb(skb);
```

Thank you!