Mars Research Group: Kernel Isolation and Beyond

https://mars-research.github.io ➦

Ziyue Pan — https://pan-ziyue.github.io

October 31, 2022

Institution of Cyberspace Security Research, Zhejiang Unversity

Table of Contents

Background

Modern operating system kernels need isolation

Rapid development Linux Kernel features over 70K commits a year.

Insecure kernel `https://www.cvedetails.com/product/47/Linux-Linux-Kernel.html` ➦

Enabled protection StackGuard [6], ASLR [16], DEP [20], SMAP [1], SMEP

Unused protection CPI [12], SafeStacks [4]

New attacks DOP [10], FUZE [24]

[1] `https://lwn.net/Articles/517475`

Kernel isolation is challenging

Execution overhead: isolation brings extra overhead.

- Hardware-based isolation is not commodity design [21, 22, 23].
- Traditional address-spaces for isolation introduces huge overhead [7].

Decomposition complexity: shared-memory kernel introduces laborious efforts.

- Isolated subsystems: seL4 [2], DCOM [3], FUSE [4]
- Virtualized kernel [3, 5, 8, 14]

[2] https://sel4.systems/Info/Docs/seL4-manual-latest.pdf
[3] https://learn.microsoft.com/en-us/openspecs/windows_protocols/ms-dcom
[4] https://github.com/libfuse/libfuse

Kernel isolation is not new

Table 1: Isolation mechanisms and overheads

Mechanisms	Example	Execution overhead
Segmentation and paging	L4 [7], Nooks [19], SIDE [18]	high
Cache-coherent cross-core invocations	FlexSC [17], MultiKernel [2]	high
Memory Protection Keys (MPKs)	Hodor [9], libmpk [15]	acceptable
EPT switching with VMFunc	Hodor [9]	acceptable
SFI and MPX	MemSentry [11]	high

Table 2: Decomposition complexity

Methodology	Example	Decomposition complexity
Clean slate designs	microkernels	high
Device driver frameworks and VMs	IOKit	high
Backward compatible code isolation	LXFI [13]	acceptable

Research Project Review

(ATC'19) LXDs: Towards Isolation of Kernel Subsystems

Cache-coherent cross-core invocations + Backward compatible code isolation

Contributions:

- (Execution overhead) asynchronous execution runtime.
- (Execution overhead) dedicated core + cross-core IPC.
- (Decomposition complexity) decomposition patterns + IDL (interface definition language).

https://www.usenix.org/conference/atc19/presentation/narayanan ➦

(ATC'19) LXDs: Towards Isolation of Kernel Subsystems

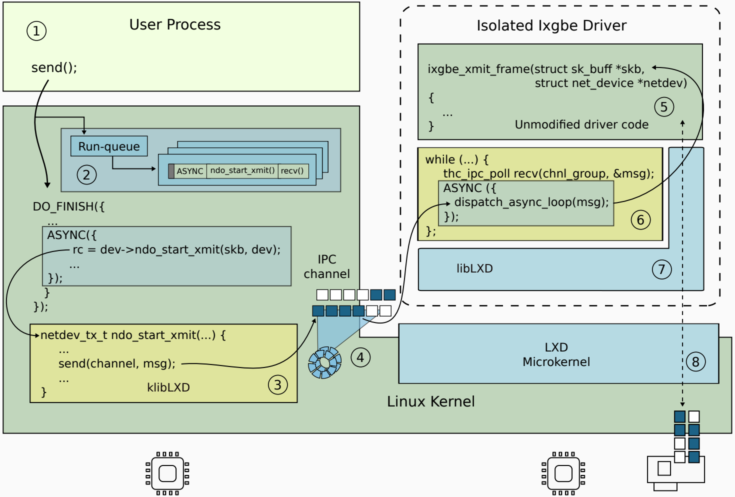
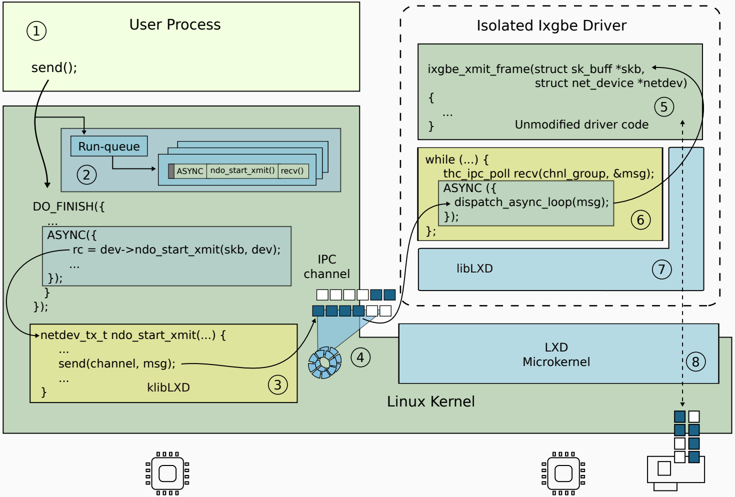


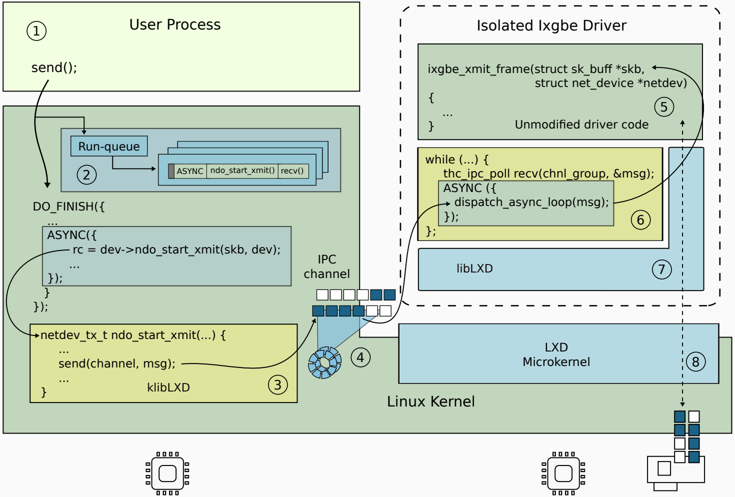
Figure 1: LXDs architecture (isolated ixgbe network driver).

(ATC'19) LXDs: Towards Isolation of Kernel Subsystems



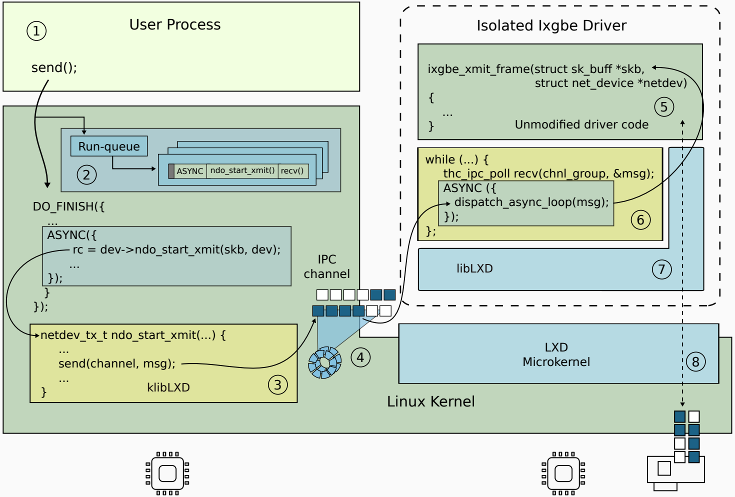
Each LXD is developed as a loadable kernel module based on VT-x.

(ATC'19) LXDs: Towards Isolation of Kernel Subsystems



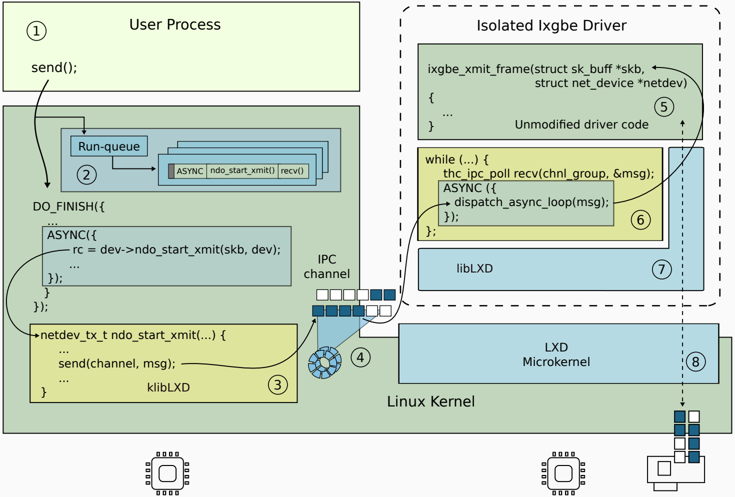
Compatibility: ⑥ glue code generated by IDL compiler and ⑦ libLXD.

(ATC'19) LXDs: Towards Isolation of Kernel Subsystems



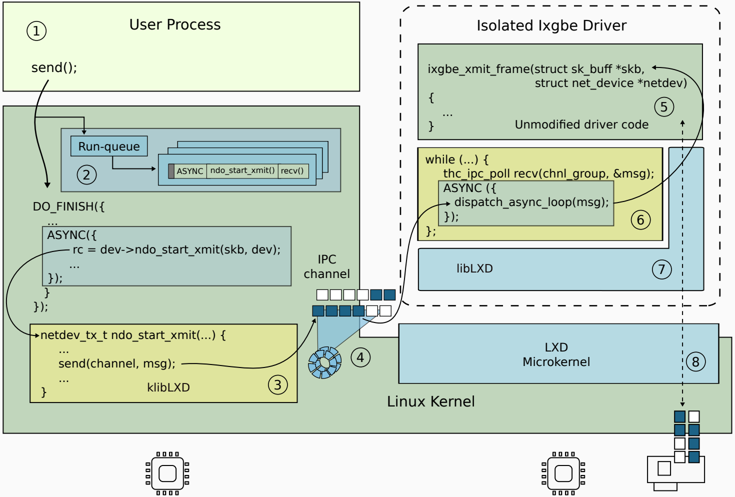
⑧ LXD microkernel creates and manages LXDs.

(ATC'19) LXDs: Towards Isolation of Kernel Subsystems



③ IDL compile generates klibLXD.

(ATC'19) LXDs: Towards Isolation of Kernel Subsystems



② Run-queue maintains async runtime; ④ supports cross-core IPC.

(ATC'19) LXDs: Towards Isolation of Kernel Subsystems

IDL: generate glue code across domain boundaries from customized grammar.

```
include <net.idl>
    module dummy() {
    require net;
}
...
module net() {
    rpc int register_netdevice(projection net_device *dev);
    rpc void ether_setup(projection net_device *dev);
    ...
}
...
projection <struct net_device> net_device {
    unsigned int flags;
    unsigned int priv_flags;
    ...
    projection net_device_ops [alloc(caller)] *netdev_ops;
}
```

https://gitlab.flux.utah.edu/xcap/lcds-idl ➦

(ATC'19) LXDs: Towards Isolation of Kernel Subsystems

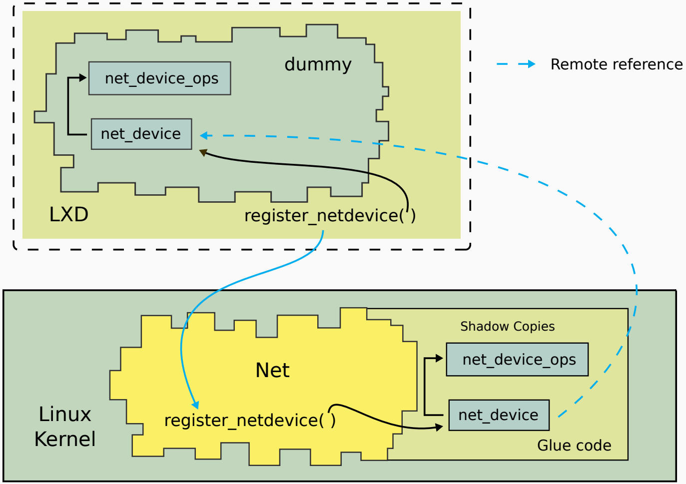


Figure 2: Shadow objects in LXDs.

(ATC'19) LXDs: Towards Isolation of Kernel Subsystems

Case studies:

- (software) dummy network and multi-queue block
- (hardware) ixbge

Evaluation:

- async runtime overhead is small
- cross-core IPC is faster than same-core IPC
- (on ixbge TX) single thread: LXD is 12% faster; multi thread: LXD is only 6%-13% of native driver
- (on ixbge RX) single thread: LXD performs similarly; multi thread: LXD is only 12%-18% of native driver

(VEE'20) Lightweight Kernel Isolation with Virtualization and VM Functions

EPT switching with VMFunc + Backward compatible code isolation

Contribution:

1. LVD: lightweight isolation (speed up LXD)
2. Isolation enforcement
   - **Data structure safety** isolated driver can only access a well-defined subset of objects and their fields
   - **Data structure integrity** isolated driver cannot change pointers used by the kernel or types of referenced objects.
   - **Function call integrity** a) can only invoke a well-defined subset of kernel functions and pass legitimate arguments; b) cannot trick the kernel into invocation of an unsafe function pointer registered as part of the driver interface.

(VEE'20) Lightweight Kernel Isolation with Virtualization and VM Functions

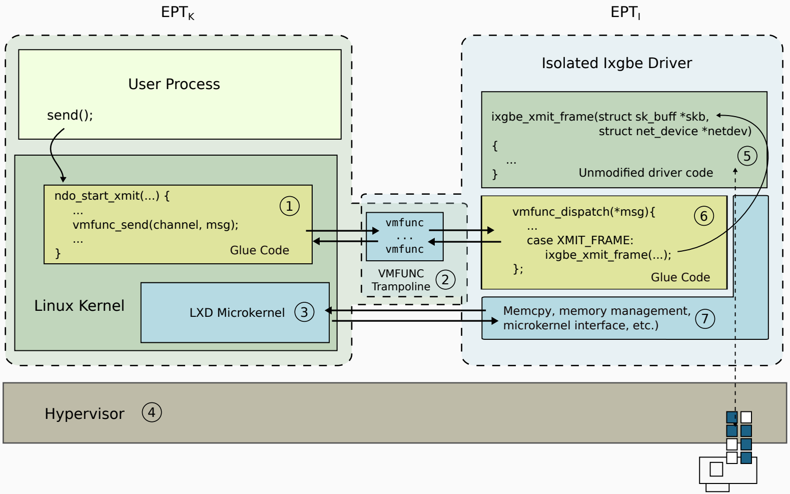
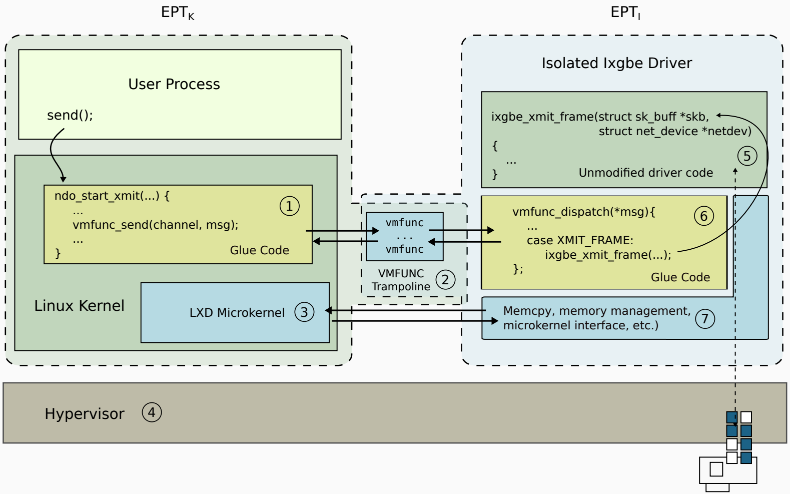


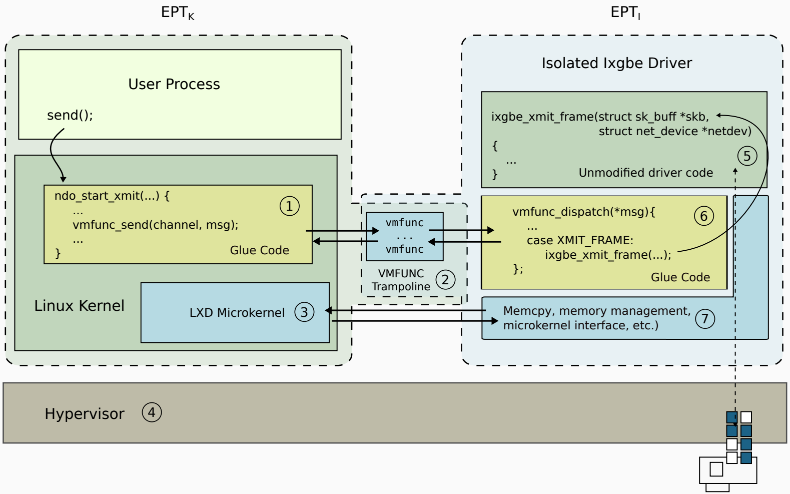
Figure 3: LVD architecture (isolated ixgbe network driver).

(VEE'20) Lightweight Kernel Isolation with Virtualization and VM Functions



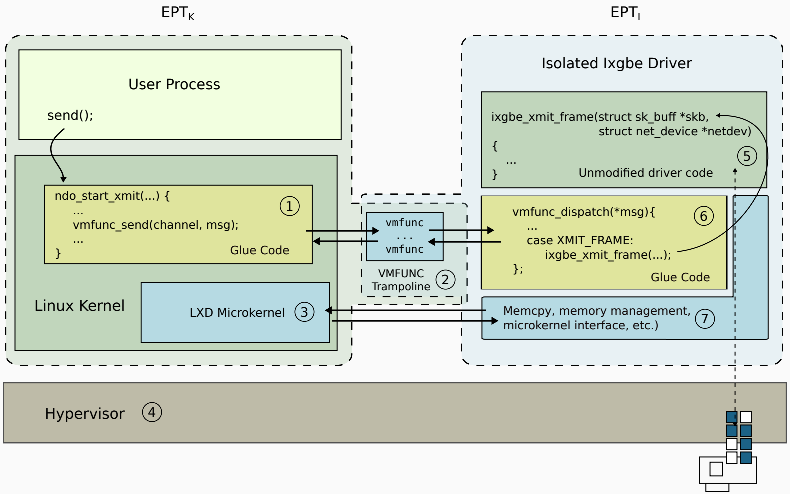
④ Modified Bareflank hypervisor [1] demotes kernel into non-root VT-x guest (EPT_k).

(VEE'20) Lightweight Kernel Isolation with Virtualization and VM Functions



When a new isolated driver is created, ③ LXD microkernel creates a new EPT_i.

(VEE'20) Lightweight Kernel Isolation with Virtualization and VM Functions



② A call-gate page with VMFunc instructions is mapped in both EPTs.

(VEE'20) Lightweight Kernel Isolation with Virtualization and VM Functions

Security invariants:

1. Virtual address spaces of isolated domains, kernel, and user processes do not overlap.
2. Isolated domains have read-only access to their page table.
3. Physical address spaces of isolated domains and the kernel must not overlap.
4. Access to sensitive state is mediated by the hypervisor.
5. General, segment, and extended state (x87 FPU, SSE, AVX, etc.), registers are saved and restored on domain crossings.

(VEE'20) Lightweight Kernel Isolation with Virtualization and VM Functions

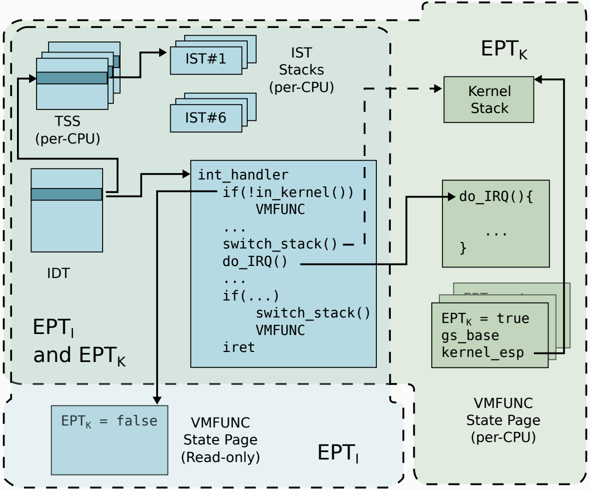


Figure 4: Exitless interrupt handling.

(VEE'20) Lightweight Kernel Isolation with Virtualization and VM Functions

Case studies:

- (software) dummy network and null block
- (hardware) ixbge

Evaluation:

- Phoronix test suite: 1% - 5% overhead (demoted kernel)
- (on dummy network TX) multi thread: 88% of native performance
- (on ixbge TX) single thread: LVD is 5% slower; multi thread: LVD is on par with native

(OSDI'20) RedLeaf: Isolation and Communication in a Safe Operating System

Language-based system + Clean slate designs

Contributions:

- Fault isolation.
- RedLeaf OS, RV6 (POSIX interface), ixgbe driver and NVMe driver.

https://www.usenix.org/conference/osdi20/presentation/narayanan-vikram ➦,
https://pan-ziyue.github.io/slides/redleaf.pdf ➦

(OSDI'20) RedLeaf: Isolation and Communication in a Safe Operating System

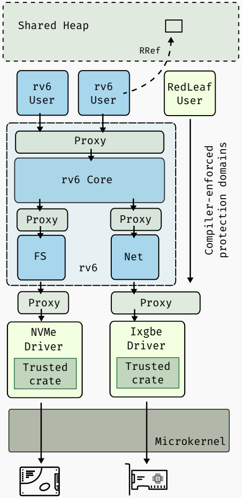


Figure 5: RedLeaf architecture

(OSDI'20) RedLeaf: Isolation and Communication in a Safe Operating System

Fault isolation principles:

- **Heap isolation**: domains never hold pointers into private heaps of other domains.
- **Exchangeable types**: types that have no pointers to private heap.
- **Ownership tracking**: track ownership of all objects on the shared heap.
- **Interface validation**: IDL enforces cross-domain interfaces.
- **Cross-domain call proxying**: IDL generates cross-domain invocation proxies.

Summary: transfer Rust semantic to OS level.

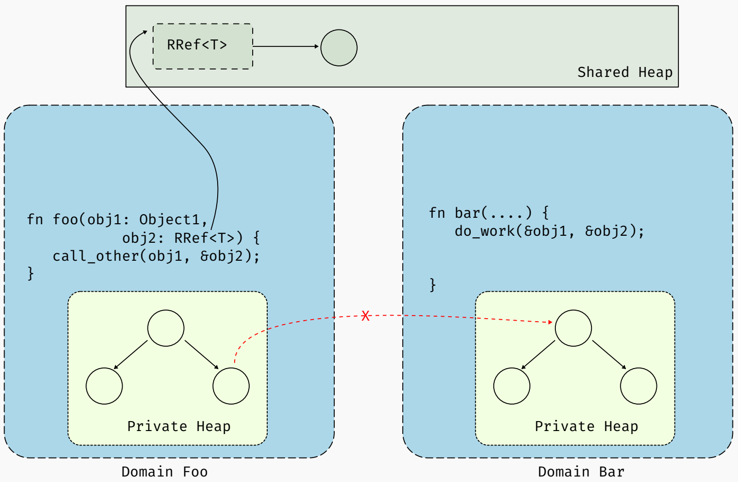


Figure 6: Heap isolation.

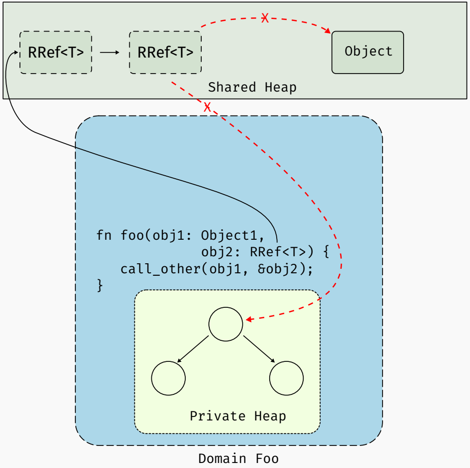


Figure 7: Exchangeable types.

(OSDI'20) RedLeaf: Isolation and Communication in a Safe Operating System

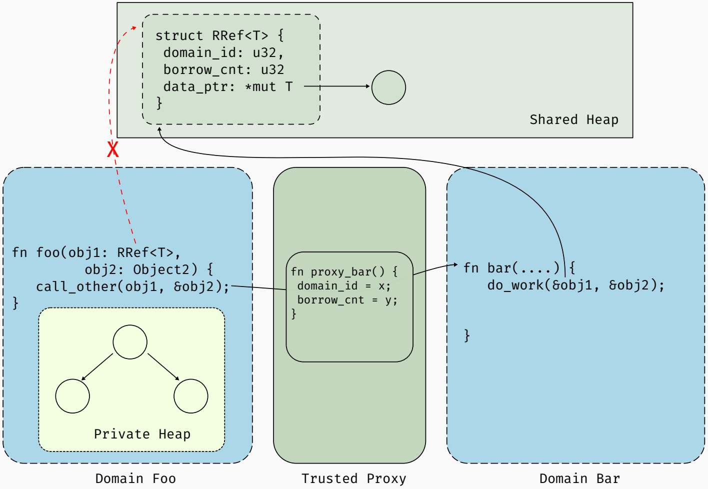


Figure 8: Ownership tracking.

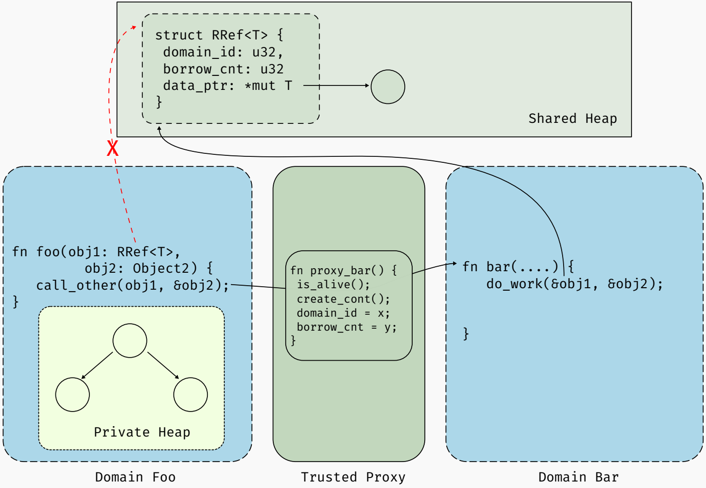


Figure 9: Cross-domain call proxying.

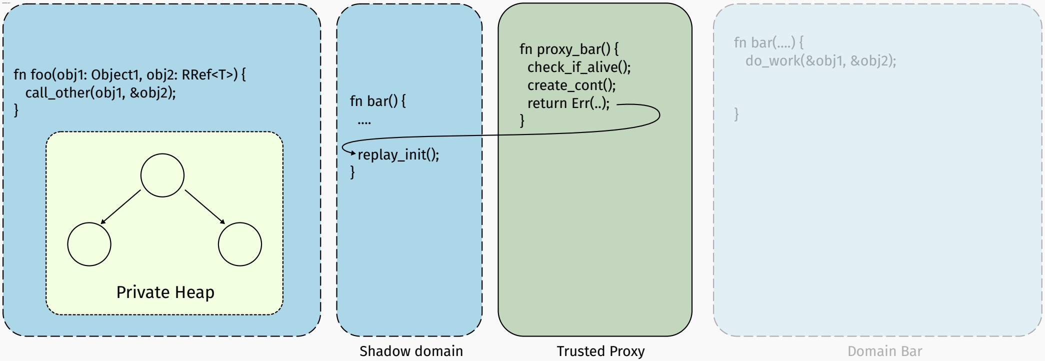


Figure 10: Device driver recovery.

(OSDI'20) RedLeaf: Isolation and Communication in a Safe Operating System

Case studies:

- (driver) ixbge and NVMe
- (application) Maglev load-balancer and Key-value store

Evaluation:

- (on ixgbe TX and RX) 32 packets batch: on par with DPDK
- (on NVMe) on par with SPDK
- (Maglev) 52% - 74% of DPDK
- (KV store) 61% - 86% of DPDK

(OSDI'22) KSplit: Automating Device Driver Isolation

Backward compatible code isolation: automated static analysis on Linux kernel for kernel isolation.

Motivating example: ixgbe involves 5,782 functions and 900,000+ object fields.

Challenges:

- **Large interface boundary**: 134+81 functions between kernel and ixgbe.
- **Complex data exchange**: only a small subset of struct fields are shared.
- **Low-level kernel/C idioms**: ptrs, tagged unions, sized and sentinel array ...
- **Concurrency primitives**: spin/mutex, atomic ops, RCU, sequential lock ...

https://www.usenix.org/conference/osdi22/presentation/huang-yongzhe ➦

(OSDI'22) KSplit: Automating Device Driver Isolation

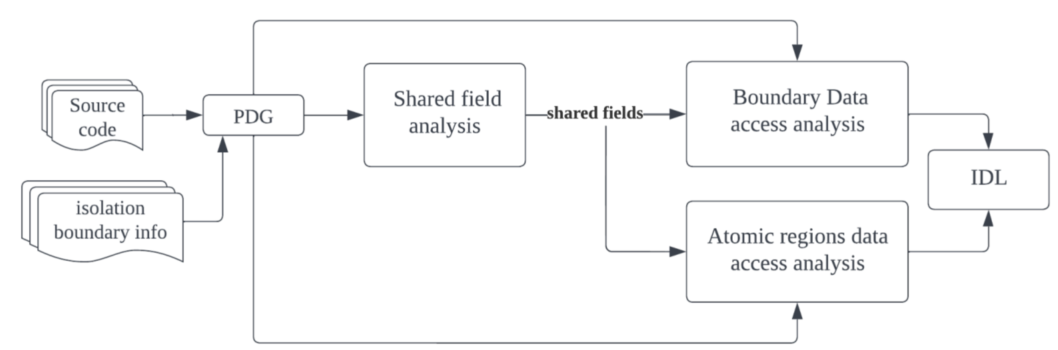


Figure 11: KSplit analysis workflow.

Beyond Isolation: a Noob's Perspective

(ATC'19) LXDs: Towards Isolation of Kernel Subsystems

ASPLOS'18 $\rightarrow$ OSDI'18 $\rightarrow$ **ATC'19**

Decoration

- Decomposition complexity: unmodified code
- IDL

(VEE'20) Lightweight Kernel Isolation with Virtualization and VM Functions

OSDI'19 → ASPLOS'19 → **VEE'20**

Decoration

- SoK of kernel isolation: execution overhead and decomposition complexity
- Isolation invariant

(OSDI'20) RedLeaf: Isolation and Communication in a Safe Operating System

ASPLOS'20 $\rightarrow$ **OSDI'20**

Decoration

- SoK of language-based OS
- Fault isolation principles

(OSDI'22) KSplit: Automating Device Driver Isolation

ASPLOS'21 $\rightarrow$ OSDI'21 $\rightarrow$ SOSP'21 $\rightarrow$ **OSDI'22**

Decoration

- Join static analysis with kernel isolation
- Kernel static analysis challenges

Takeaway

Takeaway

- Isolation challenges: execution overhead + decomposition complexity.
- All 4 papers are not smoothly accepted $\rightarrow$ a rational schedule is important.
- All 4 papers is not that "perfect" $\rightarrow$ do not get stuck in trivial points.
- Logic outline is more appealing than loosely-organized narrative.

References i

Bareflank.
Bareflank hypervisor - lightweight hypervisor sdk written in c++ with support for windows, linux and uefi.
`https://github.com/Bareflank/hypervisor`, 2022.

A. Baumann, P. Barham, P.-E. Dagand, T. Harris, R. Isaacs, S. Peter, T. Roscoe, A. Schüpbach, and A. Singhania.
The multikernel: a new os architecture for scalable multicore systems.
In *Proceedings of the ACM SIGOPS 22nd symposium on Operating systems principles*, pages 29–44, 2009.

- S. Boyd-Wickizer and N. Zeldovich.
  Tolerating malicious device drivers in linux.
  In *2010 USENIX Annual Technical Conference (USENIX ATC 10)*, 2010.
- G. Chen, H. Jin, D. Zou, B. B. Zhou, Z. Liang, W. Zheng, and X. Shi.
  Safestack: Automatically patching stack-based buffer overflow vulnerabilities.
  IEEE Transactions on Dependable and Secure Computing, 10(6):368–379, 2013.

P. Colp, M. Nanavati, J. Zhu, W. Aiello, G. Coker, T. Deegan, P. Loscocco, and A. Warfield.
Breaking up is hard to do: security and functionality in a commodity hypervisor.
In *Proceedings of the Twenty-Third ACM Symposium on Operating Systems Principles*, pages 189–202, 2011.

C. Cowan, C. Pu, D. Maier, J. Walpole, P. Bakke, S. Beattie, A. Grier, P. Wagle, Q. Zhang, and H. Hinton.
Stackguard: automatic adaptive detection and prevention of buffer-overflow attacks.
In *USENIX security symposium*, volume 98, pages 63–78. San Antonio, TX, 1998.

K. Elphinstone and G. Heiser.
From l3 to sel4 what have we learnt in 20 years of l4 microkernels?
In *Proceedings of the Twenty-Fourth ACM Symposium on Operating Systems Principles*, pages 133–150, 2013.

K. Fraser, S. Hand, R. Neugebauer, I. Pratt, A. Warfield, M. Williamson, et al.
Safe hardware access with the xen virtual machine monitor.
In *1st Workshop on Operating System and Architectural Support for the on demand IT InfraStructure (OASIS)*, pages 1–1. Boston, USA;, 2004.

- M. Hedayati, S. Gravani, E. Johnson, J. Criswell, M. L. Scott, K. Shen, and M. Marty.
  Hodor:{Intra-Process} isolation for {High-Throughput} data plane libraries.
  In *2019 USENIX Annual Technical Conference (USENIX ATC 19)*, pages 489–504, 2019.
- K. K. Ispoglou, B. AlBassam, T. Jaeger, and M. Payer.
  Block oriented programming: Automating data-only attacks.
  In *Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security*, pages 1868–1882, 2018.

K. Koning, X. Chen, H. Bos, C. Giuffrida, and E. Athanasopoulos.
No need to hide: Protecting safe regions on commodity hardware.
In *Proceedings of the Twelfth European Conference on Computer Systems*, pages 437–452, 2017.

V. Kuznetzov, L. Szekeres, M. Payer, G. Candea, R. Sekar, and D. Song.
Code-pointer integrity.
In *The Continuing Arms Race: Code-Reuse Attacks and Defenses*, pages 81–116. 2018.

Y. Mao, H. Chen, D. Zhou, X. Wang, N. Zeldovich, and M. F. Kaashoek.
Software fault isolation with api integrity and multi-principal modules.
In *Proceedings of the Twenty-Third ACM Symposium on Operating Systems Principles*, pages 115–128, 2011.

R. Nikolaev and G. Back.
Virtuos: An operating system with kernel virtualization.
In *Proceedings of the Twenty-Fourth ACM Symposium on Operating Systems Principles*, pages 116–132, 2013.

S. Park, S. Lee, W. Xu, H. Moon, and T. Kim.
libmpk: Software abstraction for intel memory protection keys (intel {MPK}).
In *2019 USENIX Annual Technical Conference (USENIX ATC 19)*, pages 241–254, 2019.

H. Shacham, M. Page, B. Pfaff, E.-J. Goh, N. Modadugu, and D. Boneh.
On the effectiveness of address-space randomization.
In *Proceedings of the 11th ACM conference on Computer and communications security*, pages 298–307, 2004.

L. Soares and M. Stumm.
{FlexSC}: Flexible system call scheduling with {Exception-Less} system calls.
In *9th USENIX Symposium on Operating Systems Design and Implementation (OSDI 10)*, 2010.

Y. Sun and T.-c. Chiueh.
Side: Isolated and efficient execution of unmodified device drivers.
In *2013 43rd Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN)*, pages 1–12. IEEE, 2013.

References x

M. M. Swift, S. Martin, H. M. Levy, and S. J. Eggers.
Nooks: An architecture for reliable device drivers.
In *Proceedings of the 10th workshop on ACM SIGOPS European workshop*, pages 102–107, 2002.

A. van de Ven.
New security enhancements in red hat enterprise linux v. 3, update 3.
Raleigh, North Carolina, USA: Red Hat, 2004.

L. Vilanova, M. Ben-Yehuda, N. Navarro, Y. Etsion, and M. Valero.
Codoms: Protecting software with code-centric memory domains.
ACM SIGARCH Computer Architecture News, 42(3):469–480, 2014.

- E. Witchel, J. Rhee, and K. Asanović.
  Mondrix: Memory isolation for linux using mondriaan memory protection.
  In *Proceedings of the twentieth ACM symposium on Operating systems principles*, pages 31–44, 2005.
- J. Woodruff, R. N. Watson, D. Chisnall, S. W. Moore, J. Anderson, B. Davis, B. Laurie, P. G. Neumann, R. Norton, and M. Roe.
  The cheri capability model: Revisiting risc in an age of risk.
  In *2014 ACM/IEEE 41st International Symposium on Computer Architecture (ISCA)*, pages 457–468. IEEE, 2014.

W. Wu, Y. Chen, J. Xu, X. Xing, X. Gong, and W. Zou.
{FUZE}: Towards facilitating exploit generation for kernel {Use-After-Free} vulnerabilities.
In *27th USENIX Security Symposium (USENIX Security 18)*, pages 781–797, 2018.