KSplit: Automating Device Driver Isolation [1]

CNS-1801534: Threat-Aware Defenses - Trent Jaeger (Penn State), Gang Tan (Penn State), Mathias Payer (Purdue/EPFL), Dongyan Xu (Purdue)
CNS-1816282: Information Flow Control Infrastructure - Trent Jaeger (Penn State), Danfeng Zhang (Penn State)

Yongzhe Huang, Vikram Naranyanan, David Detweiler, Kaiming Huang, Gang Tan, Trent Jaeger, Anton Burtsev
Penn State University, UC Irvine, University of Utah

Introduction

- Device drivers have long been and continue to be a major source of defects and vulnerabilities in modern kernels.
- Previous works on isolating device drivers: (1) significant manual effort and (2) high runtime overhead.
- Recently, some hardware features for efficient isolation have become available (e.g., vmfunc). These techniques significantly reduce the overhead of isolation. [2]
- However, isolating drivers remains hard because of the manual effort required to retrofit the code.

Motivation and Objective

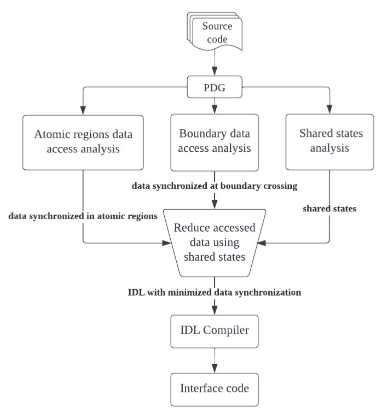
Motivation:
Reduce the manual work necessary for isolating device drivers as much as possible.

Objective: Automate most of key tasks of driver isolation using static analysis techniques and produce warnings for developers to resolve the remaining tasks.

Main Challenges

- Minimize the data that need to be synchronized across isolation boundary at **cross-domain calls and returns**.
- Correctly handle data synchronization for **kernel concurrency primitives** such as spin_lock while minimizing the amount of synchronized data.
- Correctly handle data synchronization in the presence of challenging kernel and C language idioms (e.g., pointers to complex struct hierarchies).

System Workflow

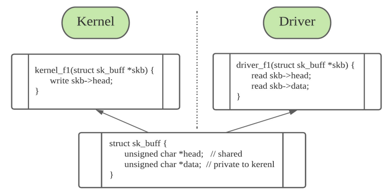


Compute Shared States

Shared state: **Shared states** are the data structure fields that are accessed by both driver and kernel through the same structure type. This information helps **limit** the amount of data that needs to be synchronized between isolated domains.

Steps:

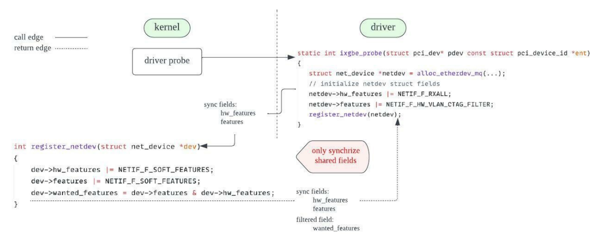
1. Compute a set of data structures accessed by both kernel and driver.
2. Identify all variables on both sides that match any of the shared data structure types, and analyze the fields accessed through these variables on each side.
3. Take the intersection between the accessed fields on both sides to obtain the shared accessed fields.



Compute shared states

Cross-domain Call Data Synchronization

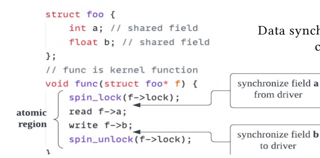
- **Data synchronization for cross-domain calls**: Compute data that needs to be synchronized at domain crossing calls and returns.
- **Data access analysis**: For each **parameter** passed across **isolation boundary**, use PDG to track the accesses to the parameter.
  - All the data **read** through the parameter during call processing is synchronized at the cross-domain call **invocation**.
  - All the data **modified** through the parameter during call processing is synchronized at the cross-domain call **return**.
- **Minimize synchronized data using shared state**: only the shared state is synchronized between the driver and kernel to minimize the overhead of cross-domain calls.



Compute and minimize data synchronized across isolation boundary

Concurrency Primitives Data Synchronization

1. Identify **atomic regions**
   a) Find atomic primitives in the PDG (e.g., spin_lock, mutex_lock).
   b) Use **control flow** in PDG to compute the code within critical sections.
2. For each atomic region
   a) Compute data **read** within the atomic region and synchronize the data from the other domain **after acquiring** the lock.
   b) Compute data **modified** within the atomic region and synchronize the data to the other domain **before releasing** the lock.



Data synchronization example for critical section

Evaluation

- KSplit reduces data synchronization by ~30% relative to prior work
- KSplit reduces the manual effort for IDL changes to <60 LOC for the ten drivers isolated and provides concrete warnings for these cases

	coretemp	nullnet	ixgbe	alx	can-raw	sb_edac	null_blk	dm_zero	msr	xhci-hcd
SLOC	562	194	27K	3K	615	2K	690	54	218	10K
Drv.→kern.	21	13	132	63	36	14	36	3	15	45
Kern.→drv.	2	11	85	26	17	1	9	2	5	27
Functions	643	1K	5K	3K	1K	912	1K	133	459	1K

(a) Complexity of driver analysis

Deep copy	31K	46K	999K	214K	153K	24K	75K	11K	24K	134K
Access analysis [32]	127	231	4K	1K	694	89	562	13	54	375
Shared analysis	92	150	3K	733	363	68	306	10	43	263
Boundary analysis	92	149	2K	724	331	68	279	10	43	197

Reduce ~30% data synchronization

(b) Total number of fields marshaled across all interface functions by each algorithm

Pointers	12K/75	19K/87	404K/1,639	82K/362	60K/153	9K/49	28K/189	4K/9	9K/41	48K/179
Unions	0/0	5/3	108/48	24/18	22/29	0/0	1/9	0/0	0/0	0/7
Critical sections	5/0	5/1	70/3	26/1	19/2	2/0	31/0	0/0	8/0	10/0
RCU	0/0	1/0	8/0	6/0	9/0	0/0	6/0	0/0	0/0	0/0
Seqlock	0/0	0/0	3/0	0/0	0/0	0/0	0/0	0/0	0/0	0/0
Atomic operations	0/0	25/1	168/40	49/32	50/0	5/0	37/2	3/0	3/0	46/8
Container of	225/4	557/2	2K/59	1K/102	749/80	419/0	627/25	73/3	68/2	1K/6

(c) Impact of shared state optimizations (private/shared)

Singleton	70/0	79/0	1,367/0	349/0	147/0	37/0	169/0	9/0	39/0	166/0
Array	0/1	4/2	92/35	35/1	22/1	9/6	13/5	0/0	0/1	1/1
String	1/0	1/0	2/0	0/0	0/0	2/0	2/0	0/0	1/0	0/0
Void	2/1	4/0	140/1	11/0	5/0	3/0	13/0	0/0	1/0	12/0
Wild pointer (non-void)	1/0	0/2	0/4	0/1	0/0	0/3	0/2	0/0	0/0	0/0

(d) Inference type semantics on shared pointers (handled/manual)

Time	17	217	546	190	135	22	490	5	7	238

(e) Analysis execution time (seconds)

Statements	70%	86%	50%	72%	79%	63%	79%	85%	77%	55%
Branches	57%	81%	48%	76%	79%	65%	91%	100%	96%	53%

(f) Test coverage

Few IDL changes required

IDL (LOC)	163	221	2K	674	470	236	306	47	109	1K
IDL change (LOC)	1	5	52	25	30	5	11	0	2	7
Drv. changes (LOC)	10	6	19	11	12	0	0	0	0	0
False positives	1	25	129	43	30	6	34	2	5	12
Ptr. misclassification	0	0	7	3	2	2	3	0	2	0
No. Warnings	1	8	65	30	35	5	18	0	5	15

Experiments on 10 automatic isolated drivers

	char/tty (77)	block (17)	net (89)	edac (13)	hwmon (67)	spi/i2c (38)	usb (53)
SLOC	1047	2535	13302	896	556	471	1340
Drv.→kern.	11	60	25	18	10	14	16
Kern.→drv.	10	16	47	4	5	3	13
Functions	546	2588	2691	839	462	772	784

(a) Complexity of driver interfaces

Pointers	15K /64	53K /310	73K /353	16K /107	10K /61	12K /71	18K /92
Unions	0/2	3/12	7/6	0/2	<1/<1	0/2	<1/4
Crit. sec.	5/<1	51/<1	25/<1	5/<1	6/<1	9/<1	9/<1
Atomic op.	<1/0	6/0	2/0	0/0	<1/0	<1/0	<1/<1
RCU	<1/0	<1/0	<1/0	0/0	<1/0	0/0	<1/<1
Seqlock	9/<1	45/2	45/11	6/0	<1/<1	4/0	10/<1
Container of	145/4	833/3	1K/9	338/2	133/2	207/2	215/3

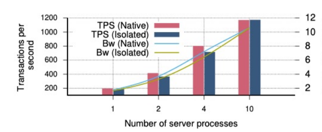
(b) Impact of shared state optimizations (private/shared)

Singleton	53/0	26/0	303/0	84/0	56/0	66/0	81/0
Array	5/2	27/15	44/20	22/6	2/<1	4/2	4/1
String	<1/0	3/0	<1/0	2/0	<1/0	<1/0	<1/0
Void	5/<1	18/0	12/1	3/0	1/<1	2/<1	6/<1
Non-void wild pointer	0/<1	0/2	0/3	0/3	0/<1	0/<1	0/2

Complexity metrics across several subsystems

	Null	Integer	Array	String	Void	Union
Bytes	0	8	32 * 8	256	4096	24 + 32
Cycles	502	532	690	1310	919	710

Overhead of marshaling different data structures



Memcached performance

Reference	ixgbe		skx_edac
	nullnet	alx	sb_edac
Shared rpcs	11	73	13
Shared rpcs IDL_{Δ}	+0/-51	+12/-29	+1/-1
Shared rpcs $Annotat._{\Delta}$	0	+3/-3	0
New IDL	77	36	0

IDL similarity among similar drivers

Conclusion

- Commodity CPUs are converging on a set of practical hardware mechanisms capable of providing support for low-overhead isolation
- The complexity of driver isolation becomes the main challenge for enabling isolation in commodity systems.
- KSplit takes a step forward by enabling isolation of unmodified device drivers in the Linux kernel.

References

[1], Huang, Y., Narayanan, V., Detweiler, D., Huang, K., Tan, G., Jaeger, T., & Burtsev, A. (2022, July). KSplit: Automating Device Driver Isolation. In *Proceedings of the 16th USENIX Symposium on Operating Systems Design and Implementation (OSDI'22)*. Conditionally accepted.

[2] Narayanan, V., Huang, Y., Tan, G., Jaeger, T., & Burtsev, A. (2020, March). Lightweight kernel isolation with virtualization and VM functions. In *Proceedings of the 16th ACM SIGPLAN/SIGOPS International Conference on Virtual Execution Environments (VEE'20)* (pp. 157-171). Awarded **Best Paper** of the conference.