CS260 – Advanced Systems Security

Isolation

April 28, 2025

Problem

- Software may be compromised
  - Hijacked to do whatever an adversary wants
  - E.g., ROP and Data-oriented programming

Scenario

- Have a program
  - Consists of
    - Critical functionality – high secrecy and integrity
    - Untrusted user input handling
- What could go wrong?

Attack

- Attacker may compromise complex handling of untrusted user inputs
- To compromise critical functionality
  - Or just hijack the process
  - Which may have high privilege

What Should We Do?

- As defenders…

Isolation

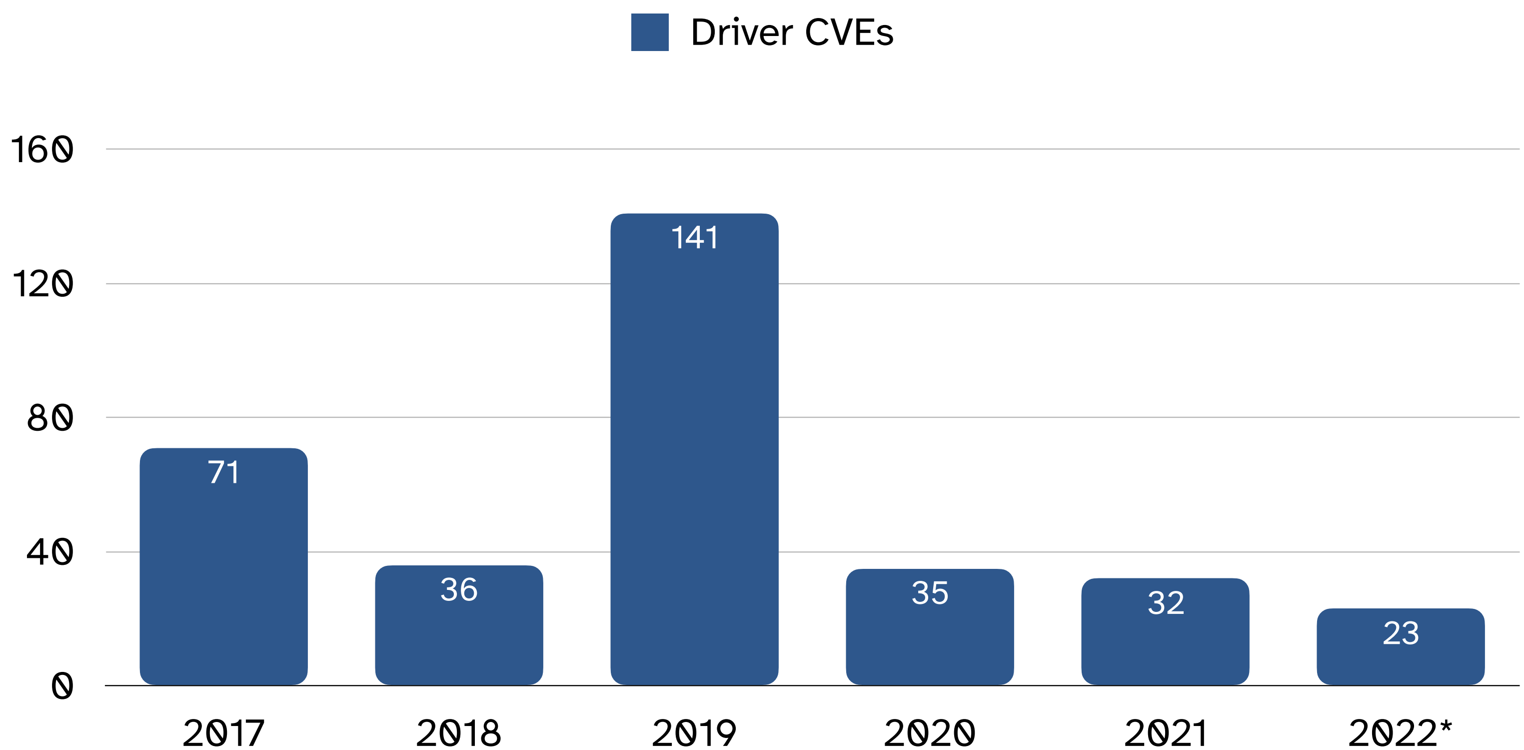
- Separate the high privilege functionality from the untrusted functionality
- Prevent the untrusted functionality from compromising the privileged functionality
- Sometimes called
  - Privilege separation
  - In-process isolation

- How would this work?

Common Goal: Driver Isolation

Driver vulnerabilities

- 16-50 % of all Linux kernel CVEs



Lots of Efforts

LXDs: Towards Isolation of Kernel Subsystems

VirtuOS: an operating system with kernel virtualization

Ruslan Nikolaev, Godmar Back
rnikola@vt.edu, gback@cs.vt.edu
Virginia Tech
Blacksburg, VA

Tolerating Malicious Device Drivers in Linux

Silas Boyd-Wickizer and Nickolai Zeldovich
MIT CSAIL

Lightweight Kernel Isolation with Virtualization and VM Functions

MIT/LCS/TR-196

FINAL REPORT OF THE MULTICS KERNEL DESIGN PROJECT

Decaf: Moving Device Drivers to a Modern Language

Matthew J. Renzelmann and Michael M. Swift
University of Wisconsin–Madison
{mjr, swift}@cs.wisc.edu

Microdrivers: A New Architecture for Device Drivers

Vinod Ganapathy, Arini Balakrishnan, Michael M. Swift

Nooks: An Architecture for Reliable Device Drivers *

The SawMill Multiserver Approach

Alain Gefflaut, Trent Jaeger, Yoonho Park, Jochen Liedtke *, Kevin J. Elphinstone †, Volkmar Uhlig †

How's It Done?

- Separate memory space

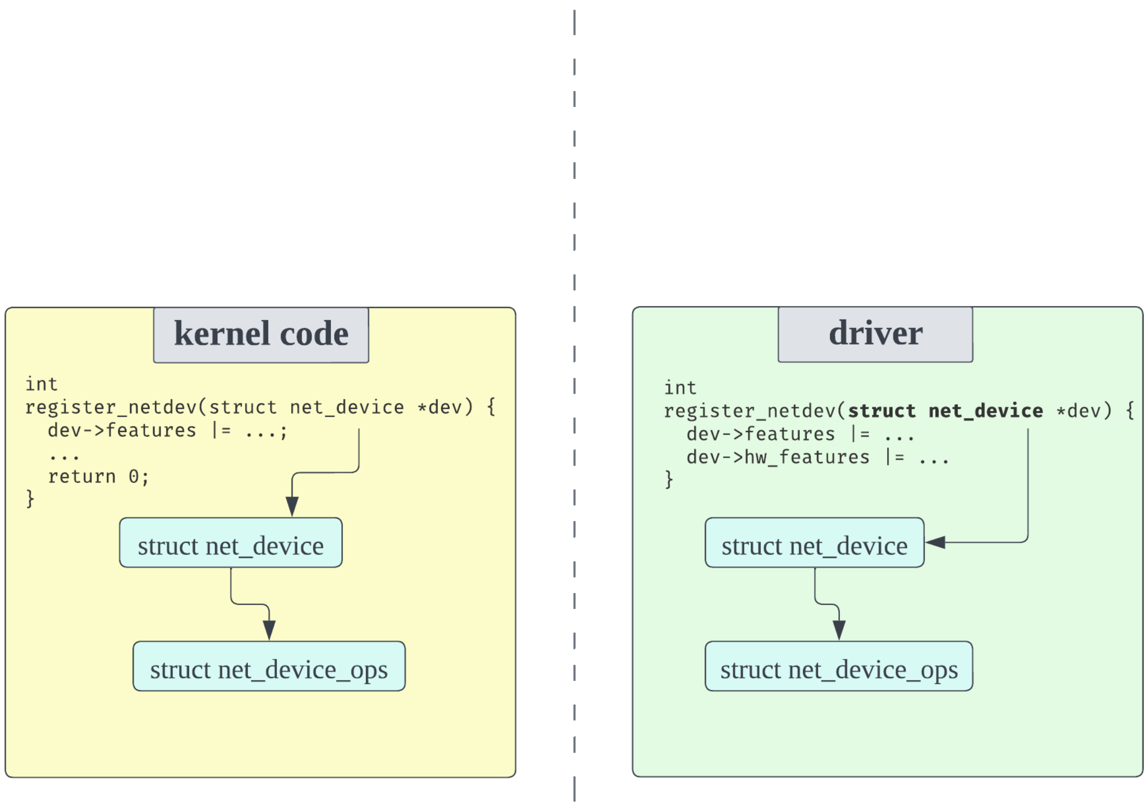
kernel code

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

driver

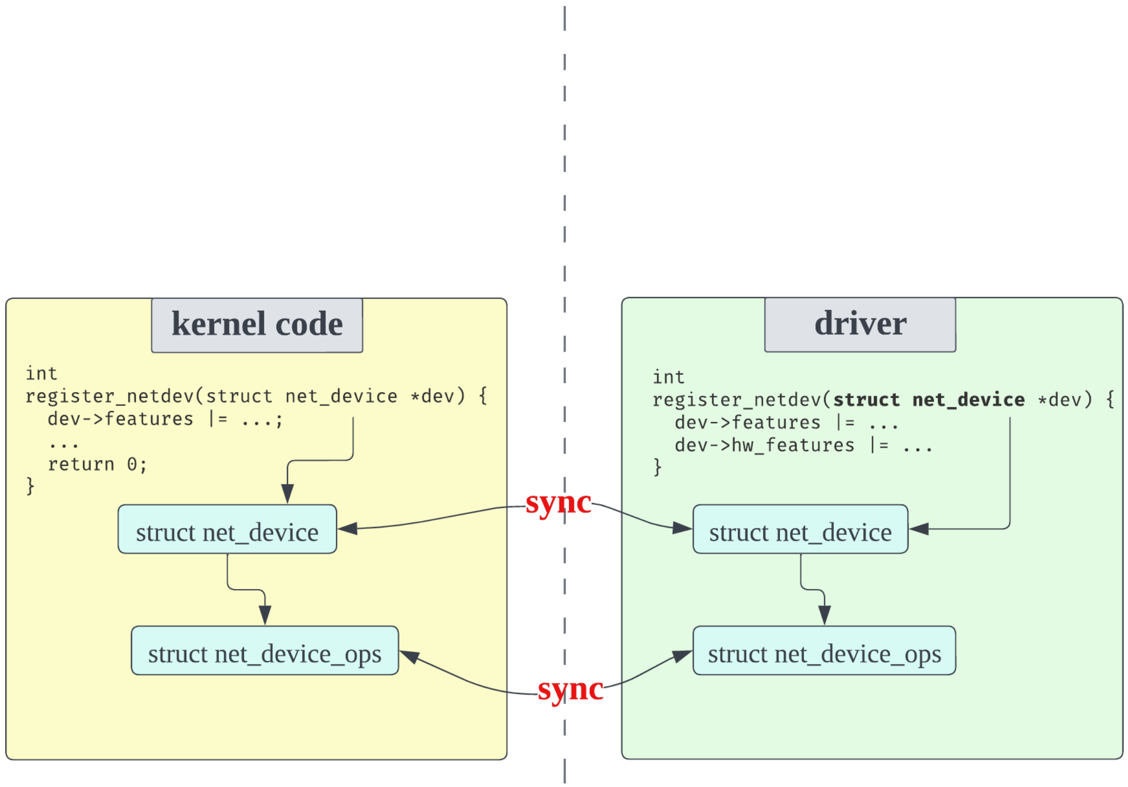
```
int
register_netdev(struct net_device *dev) {
  dev->features |= ...
  dev->hw_features |= ...
}
```

How’s It Done?



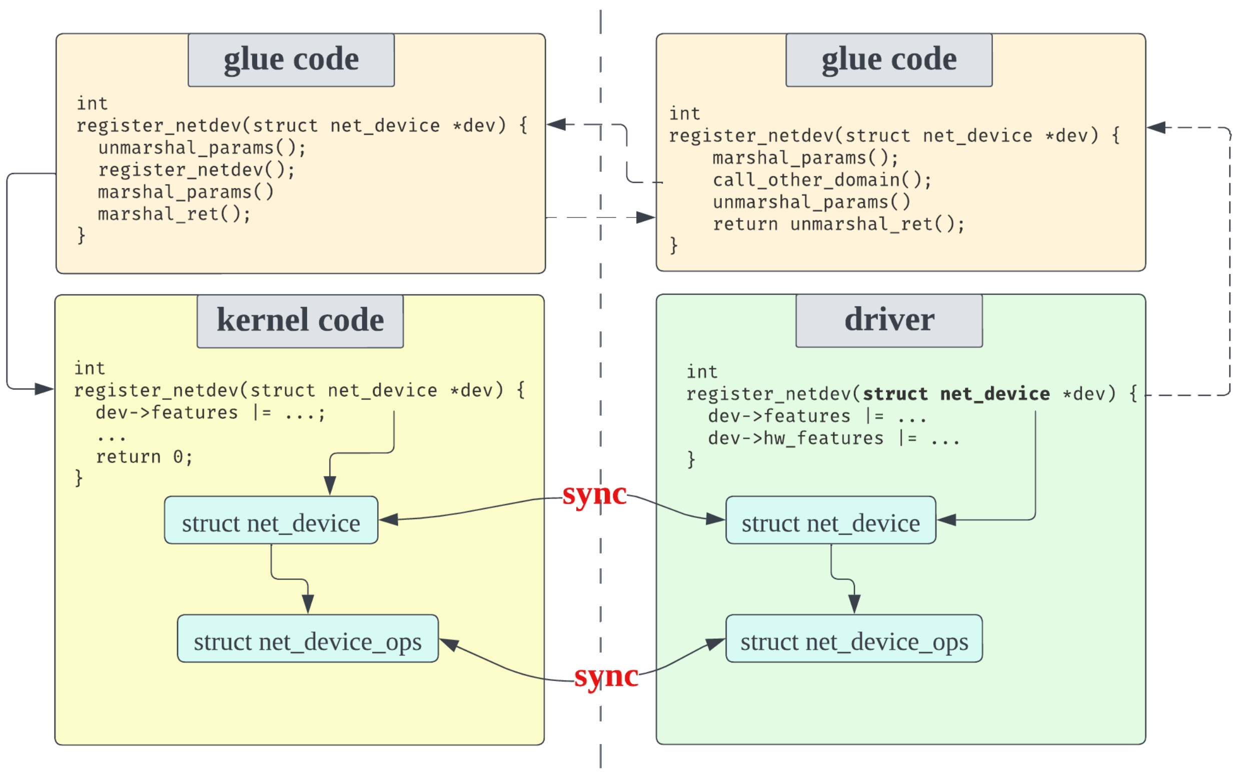
- Separate memory space
  - Two copies of object hierarchies

How's It Done?



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

How’s It Done?



- 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

Implement Isolation

- How should isolation be implemented?
  - E.g., Separate processes

Overhead of Naïve Isolation

Isolation performance

- Paging (834 cycles)
- Recent CPU mechanisms
  - VMFUNC - 396 cycles
  - MPK 11-260 cycles
  - Save/restore general/extended regs, pick a stack, etc.

Requirements

- What are our requirements for isolation?

Requirements

- What are our requirements for isolation?
  - Memory accesses limited to own region
  - Good performance
- That enough?

Requirements

- What are our requirements for isolation?
  - Memory accesses limited to own region
  - Good performance

- That enough?
- Not really – all interactions between untrusted and privileged regions are still suspect
  - Limit control flows between regions (like gates)
  - Also, must worry about data conveyed between regions
- Just moved the attack surface

Limitations and Opportunities of Modern Hardware Isolation Mechanisms

Xiangdong Chen, Zhaofeng Li, University of Utah;
Tirth Jain, Maya Labs;
Vikram Narayanan, Anton Burtsev, University of Utah

Lightweight Fault Isolation: Practical, Efficient, and Secure Software Sandboxing

Zachary Yedidia

Stanford University

Basic Motivation

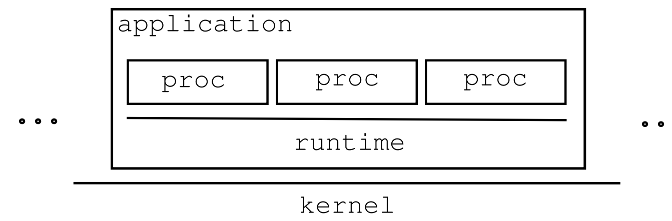
Today's systems increasingly run untrusted code.

- Web browsers (JavaScript, WebAssembly).
- Cloud machines and serverless (VMs, containers, WebAssembly).
- Kernels (eBPF).
- Smart contracts (WebAssembly, EVM).

These applications demand lightweight sandboxing in a single address space.

Goal: enforce that untrusted programs

- cannot read/write external memory.
- cannot directly perform system calls.



Trade-Offs

Native

```
source language
    ↓ .c
compile
    ↓ .elf
run (unsafe)
```

WebAssembly-LLVM

```
source language
    ↓ .c
compile
    ↓ .wasm
verify (~2K LoC)
    ↓ .wasm
compile (~2M LoC)
    ↓ .elf
run (safe??)

~20% overhead
~43,000 sandboxes
```

WebAssembly-Cranelift

```
source language
    ↓ .c
compile
    ↓ .wasm
verify (~2K LoC)
    ↓ .wasm
compile (~200K LoC)
    ↓ .elf
run (safe?)

~70% overhead
~43,000 sandboxes
```

Problem: tradeoff between performance and security.

Verification

SFI (from SOSP '93): Don't trust the compiler.

Classic SFI

```
source language
   ↓ .c
compile
   ↓ .s
instrument
   ↓ .elf
verify
   ↓ .elf
run
```

From Wasm to LFI

WebAssembly-LLVM

```
source language
   ↓ .c
compile
   ↓ .wasm
verify (~2K LoC)
   ↓ .wasm
compile (~2M LoC)
   ↓ .elf
run (safe??)
```

~20% overhead
~43,000 sandboxes

WebAssembly-Cranelift

```
source language
   ↓ .c
compile
   ↓ .wasm
verify (~2K LoC)
   ↓ .wasm
compile (~200K LoC)
   ↓ .elf
run (safe?)
```

~70% overhead
~43,000 sandboxes

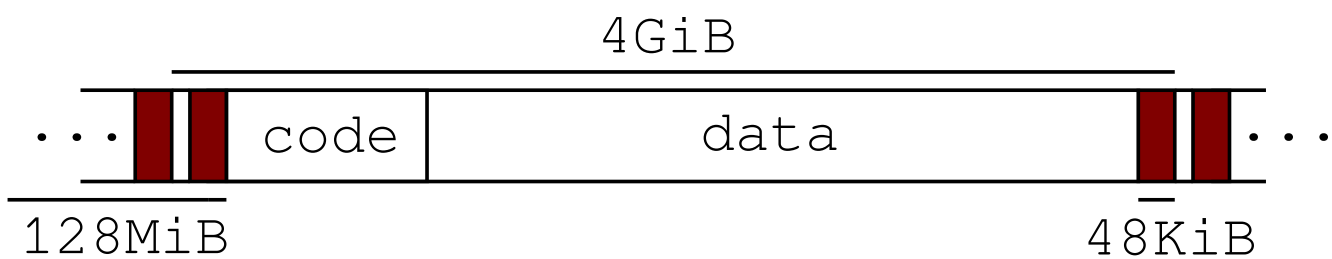
LFI (ARM64)

```
source language
   ↓ .c
compile (~2M LoC)
   ↓ .s
instrument (~2K LoC)
   ↓ .elf
verify (~400 LoC)
   ↓ .elf
run (safe)
```

~7% overhead
~65,000 sandboxes

Presenting Lightweight Fault Isolation: **low overhead**, **secure**, **scalable**, **simple**.

LFI Approach



Principles:

- Use 4GiB sandboxes combined with instructions to operate on 32-bit values.
- Works without modification to existing compilers.
  - → just reserve `x18` and `x21` when compiling.
  - → works with any language and any optimization pipeline.
- Every address in the sandbox is a valid branch target (no aligned bundles).
  - → helps simplicity, code size, running time, and Spectre-safety.

Leverages ARM Architecture Features

Important ARM64 features for SFI:

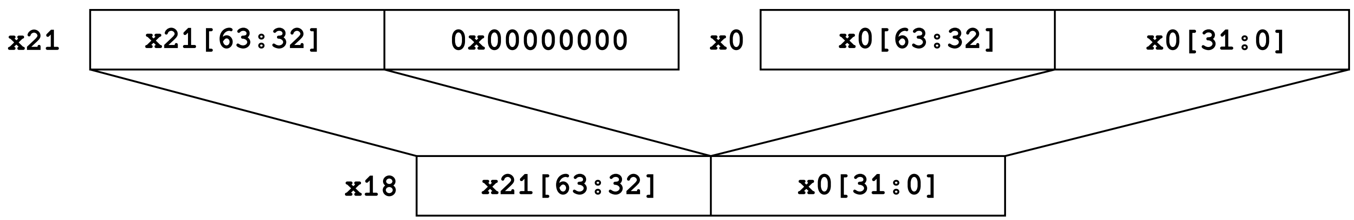
- Fixed-width encoding: misalignment traps.
  → Consistent disassembly without aligned bundles.
- 32 64-bit registers (`x0`-`x30`, `sp`).
- Stack pointer register (`sp`).
- Dedicated return address register (`x30`).
  → Easy to reserve registers.
- 32-bit register subsets (`w0`-`w30`, `wsp`).
- A 32-bit addressing mode.
  → Fast operations for 4GiB sandboxes.

Check Implementation

```
mov x18, x0 // unsafe
```

How to safely modify `x18`?

- `x21`: sandbox base address (aligned to 4GiB).




```
add x18, x21, w0, uxtw // safe
ldr x1, [x18]          // safe
```

If `x0` contained a valid address, the `add` is a `mov`.

Otherwise, sandbox has non-escaping undefined behavior (pointer overflow).

Instrumentation

Original	Sandboxed
`ldr x1, [x0]`	`add x18, x21, w0, uxtw` `ldr x1, [x18]`
`br x0`	`add x18, x21, w0, uxtw` `br x18`

Instrumenter performs transformations; verifier is convinced of their safety.

Same invariant for `sp` and `x30`.

```
ret x30           // safe
ldr x1, [sp, #8]  // safe
```

`ldr x30, [sp]`	`ldr x30, [sp]` `add x30, x21, w30, uxtw`

Optimization

Key Optimization: we can perform the guard inside a load/store addressing mode.

Original code	Sandboxed equivalent	Cycles of overhead
`ldr rt, [xN]`	`ldr rt, [x21, wN, uxtw]`	0
`ldr rt, [xN, #i]`	`add w22, wN, #i` `ldr rt, [x21, w22, uxtw]`	1
`ldr rt, [xN, #i]!`	`add xN, xN, #i` `ldr rt, [x21, wN, uxtw]`	1
`ldr rt, [xN], #i`	`ldr rt, [x21, wN, uxtw]` `add xN, xN, #i`	1

(other addressing modes omitted for brevity)

Verifier

```
add x0, x1, x2    // safe
b foo             // safe
svc #0 (syscall) // unsafe
br x0             // unsafe
ldr x1, [x0]      // unsafe
mov x18, x0       // unsafe
```

Unsafe instruction → rejected by verifier.

Special/reserved register (same idea from the original 1993 SFI project):

- `x18`: always contains a valid sandbox address.

```
ldr x1, [x18]  // safe
```

Take Away

- One way to protect our programs from attack
  - Isolate critical functionality from untrusted functionality
- But, balancing security and performance …
  - And programmability and usability
  - … is difficult
- Examined modern hardware isolation features
  - Not perfect balance
- And recent work in software fault isolation

Questions