Transparent Fault Isolation using Dynamic Compilation

Peter Feiner
peter@cs.toronto.edu

Angela Demke Brown
demke@cs.toronto.edu

Ashvin Goel
ashvin@eecg.toronto.edu

University of Toronto

Problem: Isolating Faults in Drivers

Drivers Cause Most Kernel Panics

- Drivers tend to have more bugs than the kernel in Windows and Linux. In such monolithic operating systems, drivers are not isolated from the kernel, so drivers cause most panics.

Fault Isolation

- The general technique of restricting errant writes and branches, protecting memory and control flow.

Transparently Isolating Drivers

- Existing isolation techniques require rewriting or recompiling drivers. Process-based isolation is only transparent when drivers use no global data structures. Moreover, process-based isolation is expensive for the frequent and fine-grained interaction between drivers and the kernel.

Comparison of Fault Isolation Techniques

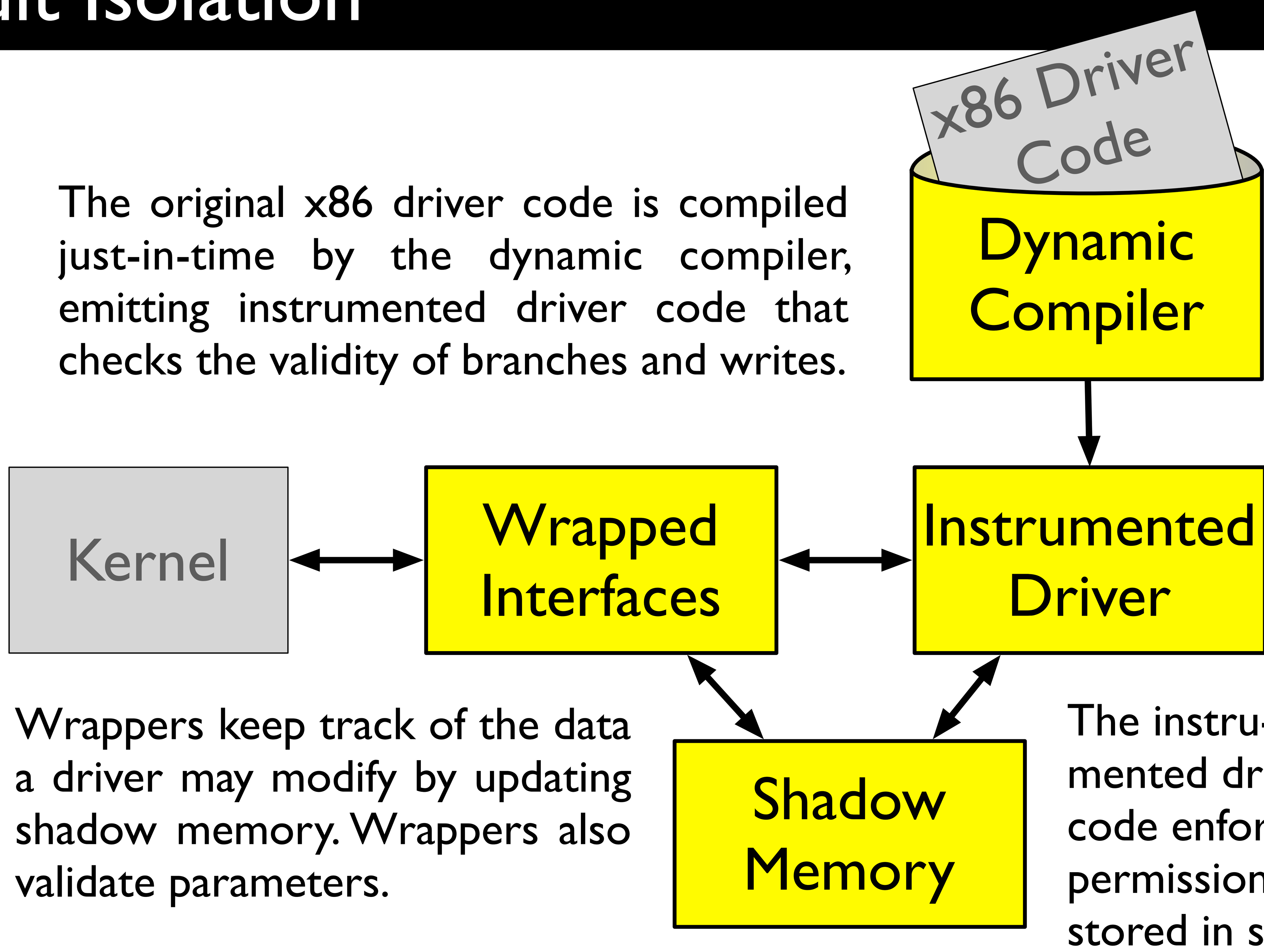
Protects:	Memory	Control Flow	Arbitrary Binaries
Transparent Fault Isolation	✓	✓ limited	✓
Program Shepherding	✗	✓ limited	✓
BGI	✓	✓	✗
XFI	✓	✓	✗
Nooks	✓	✓ limited	✗

Solution: Transparent Fault Isolation

Main Idea

- Dynamically add permission-checking instructions to existing x86 code.
- Track permissions with thin wrappers around kernel-driver interfaces.
- Solution is well suited for frequent, fine-grained interaction, like drivers!

The original x86 driver code is compiled just-in-time by the dynamic compiler, emitting instrumented driver code that checks the validity of branches and writes.



Wrappers keep track of the data a driver may modify by updating shadow memory. Wrappers also validate parameters.

The instrumented driver code enforces permissions stored in shadow memory.

Dynamic Compilation

- Prefaces **`writes`** with <u>**`checks`**</u>:

```
cmp [%eax]'s shadow, $WRITE
jne error
mov [%eax], $1234
```

  where <u>**`$WRITE`**</u> is constant.
- Checks execution permissions before linking code cache fragments.
- The dynamic compiler is protected implicitly!

Research Challenges

Dynamic Compilation

- We cannot statically identify writes to local variables, unlike schemes that control code generation. We are investigating range-based heuristics to safely elide such checks.
- There is no suitable dynamic compiler for drivers, so we are porting DynamoRIO to the Linux kernel. Interposing on interrupt handlers without monitoring all kernel execution is a challenge.

Shadow Memory

- Giving each extension its own shadow memory has several advantages: no *a priori* grouping, no inter-extension race conditions. How do we limit memory use? How do we garbage collect shadow memory?
- Efficient user-space implementations allocate large blocks of virtual memory. Shadow memory allocation in the kernel is tricky because of the absence of swappable virtual memory.